Universidad Politécnica de Cartagena

Departamento de Tecnologías de la Información y las Comunicaciones

IMPLEMENTACIÓN DE MECANISMOS DE CONTROL DE PONTENCIA A NIVEL MAC EN REDES DE SENSORES INALÁMBRICAS BAJO TINY/OS

Alexis Alcalá Galán

Director:
Javier Vales Alonso

Septiembre de 2007

Indice

Índice 3
Resumen 5
Objetivos 6
Capítulo 1. Redes de Sensores 8
Capítulo 2. Los dispositivos 12
2.1 Hardware de los dispositivos 13
2.1.1 Baterías 13
2.1.2 Consideraciones de la antena 13
2.1.3 Almacenamiento de los datos 15
2.1.4 Sensor boards 15
2.1.5 Conector de Ampliación 17
2.2 Plataforma de Programación 19
2.2.1 Introducción 19
2.2.2 Programación de los dispositivos 20
2.2.3 ISP 20
2.2.4 Botón de Reset 20
2.2.5 Alimentación de la placa 21
2.3 Lenguajes de programación para los dispositivos 22
2.3.1 Introducción 22
2.3.2 nesC 23
2.3.2.1 Introducción 23
2.3.2.2 Estructura de un componente 25
2.3.2.3 Tipos de datos 27
2.3.2.4 Tipos de funciones 28
2.4 Sistemas Operativos para los dispositivos 30
2.4.1 Introducción 30
2.4.2 TinyOS 32
2.4.2.1 ¿Qué es TinyOS? 32
2.4.2.2 Componentes primitivos de TinyOS 33
2.4.2.3 Estructura de TinyOS 38
Capítulo 3. Protocolos MAC para Redes de Sensores 40
3.1 Introducción 40
3.2 Clasificación de protocolos MAC 42

3.3 Elección del protocolo MAC 44
3.4 Protocolo S-MAC 46
3.4.1 Introducción 46
3.4.2 Mecanismo de control de potencia de transmisión 48
3.5 Aplicación “SMACTest” 51
Capítulo 4. Modificación del código 57
4.1 Introducción 57
4.1.1 Cygwin 60
4.2 Modificación de la capa MAC para implementar el mecanismo TPC 61
4.2.1 Lectura de la relación Señal a Ruido del paquete CTS recibido 62
4.2.2 Modificación de la potencia de salida 65
4.3 Modificación de la potencia de salida de los paquetes de control 71
4.4 Depuración 72
Capítulo 5. Mediciones 74
5.1 Introducción 74
5.2 Problemas de las mediciones 75
5.2.1 Primera solución 75
5.2.2 Segunda solución 76
5.2.3 Tercera solución 78
5.2.4 Fuentes de Alimentación 80
5.3 Mediciones 82
Capítulo 6. Conclusiones 86
Acrónimos y Abreviaturas 87
Bibliografía 88

Resumen

Tras haber dedicado tantos esfuerzos en crear un mecanismo para controlar la potencia de un nodo sensor, es necesaria la puesta en práctica del mismo y la comprobación de buen funcionamiento. Es lo que se han intentado en este proyecto: tras haber modificado el protocolo Sensor MAC para incorporarle el mecanismo de Control de Potencia Transmitida, se ha medido la potencia empleada en la ejecución de los nodos sensores. Hay que tener presente que en [8], se anuncia un ahorro menor en S-MAC que en otros protocolos a nivel MAC y, en la práctica, de desconoce el verdadero funcionamiento de este intento de ahorrar energía.

En este proyecto, intentaremos poner en práctica lo que se ha desarrollado teóricamente para hablar en términos prácticos de la eficiencia del mecanismo TPC.

Objetivos

El objetivo de este proyecto es obtener un uso eficiente de los recursos del canal radio, ya que está demostrado que el mayor gasto energético en los sensores se debe a la potencia gastada en la etapa radio. Para ello, se pretende emplear el mecanismo de Control de Potencia de Transmisión (TPC) en la que los datos se envían a la mínima potencia requerida y la secuencia RTS-CTS-ACK a la potencia nominal (máxima). Con este mecanismo, se usará el protocolo Sensor MAC (S-MAC) en la capa MAC y el Sistema Operativo TinyOS. Los nodos sensores de la Red Inalámbrica de Sensores serán los Mica2.

Para cumplir con nuestros objetivos, se tendrá que modificar la potencia a la que se transmite los paquetes de control (RTS,CTS,ACK) para que lo hagan a la máxima potencia de transmisión. Una vez se ha implementado el TPC, se harán pruebas para comprobar el ahorro energético, conociendo de antemano que, para S-MAC, el ahorro es bastante menor que en otros protocolos de la capa MAC. Las pruebas se realizarán mediante el depurador implementado de S-MAC (a través del hyperterminal) y midiendo, mediante un osciloscopio, la cantidad de energía empleada en el funcionamiento de los nodos y a lo largo de un gran segmento de tiempo.

Con este proyecto, se pretende demostrar que existe un pequeño ahorro energético. Y, según los resultados, conocer si merece la pena implementarlo en nuestros nodos sensores.

Capítulo 1: Redes de Sensores

En los últimos años, se ha producido un gran incremento de la presencia de las comunicaciones inalámbricas en la sociedad. A nadie le sorprende hoy contemplar pequeñas redes inalámbricas desde un teléfono con tecnología *bluetooth* con un dispositivo manos libres, o de una PDA con un PC o cualquier tecnología inalámbrica similar. Por otra parte, cada vez más, en los hogares se dispone de conexión a internet mediante tecnologías como *WiFi* (IEEE 802.11x). En definitiva, no es de extrañar que cada vez más, las investigaciones y nuevos desarrollos se centren en tecnologías inalámbricas.

Con creciente frecuencia, comienzan a aparecer sistemas y servicios basados en tecnologías inalámbricas que mejoran los procedimientos que, tradicionalmente, requerían una interacción directa por parte del ser humano. Uno de los más claros ejemplos de este tipo de sistemas son las redes de sensores inalámbricos (*Wireless Sensor Networks* o WSN). El uso de este tipo de redes presenta un futuro emocionante en el que la forma en la que nos relacionamos con nuestro entorno puede cambiar de forma sustancial. Este tipo de redes abre un horizonte nuevo de aplicaciones y servicios conscientes de su entorno en el que las posibilidades son infinitas.

Las redes de sensores inalámbricos están formadas por un conjunto de nodos autónomos capaces de realizar tareas tales como monitorización ambiental o industrial. Existe una amplia variedad de sensores, así como una amplia variedad de situaciones para “sensorizar”; los hay de temperatura, de presión, de velocidad, de luminosidad, de aceleración, de volumen, de presencia y así hasta un sinfín de actividades que potencialmente requieran monitorización. Si además de la utilidad como sensores les añadimos la posibilidad de formación de redes *ad-hoc* mediante tecnología inalámbrica, se entiende fácilmente el motivo del gran auge que están teniendo estos dispositivos en la actualidad.

Una de las ventajas que tiene este tipo de redes es la de poder operar durante largos períodos de tiempo sin intervención humana. Por ejemplo, podría instalarse una red de sensores en entornos que son demasiado peligrosos para ser controlados por humanos (por ejemplo, monitorizar la actividad sísmica de un volcán) o en entornos en los que sería demasiado costoso ser mantenidos por humanos (véase, por ejemplo, exploración espacial). Por otra parte, podrían instalarse en un edificio como medida de seguridad (detectores de presencia) o en un invernadero para controlar la humedad. Las aplicaciones son muy diversas pero todas ellas tienen características comunes: tecnología inalámbrica, bajo coste y tamaños reducidos.

Además de sus prometedoras posibilidades, este tipo de redes presentan a los investigadores una serie de retos debido a sus especiales características: número muy elevado de dispositivos, limitadas prestaciones, trabajo en grupo en pro de una tarea común, entornos con condiciones adversas...

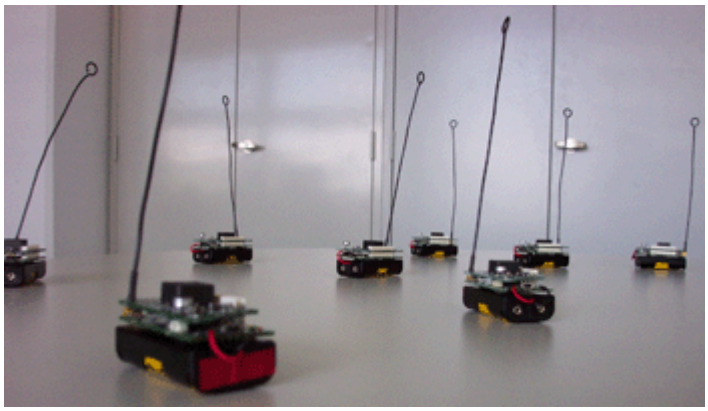

Figura 1: Red inalámbrica de sensores

Debido a su similitud, las redes de sensores heredan una serie de propiedades importantes de las redes *ad-hoc*: control descentralizado, ausencia de infraestructura de red, comunicaciones *broadcast*, canal de transmisión compartido, topologías de corta duración, redes multisalto... Además, cumplen una serie de condiciones propias tales como **tolerancia a fallos** (que toda la red no dependa de un nodo y que si uno cae, no afecte de sobremanera a la red), **escalabilidad** (el número de sensores en funcionamiento puede variar dependiendo del uso; dependiendo de la situación se podrían necesitar cientos, miles, e incluso millones de sensores), y un **mínimo coste de producción** (dado que hablamos de redes con un gran número de nodos, el coste por nodo debería ser bastante bajo). Una de las posibles aplicaciones de estas redes sería la detección de incendios, en este caso, la arquitectura de red sería similar a la del siguiente gráfico:

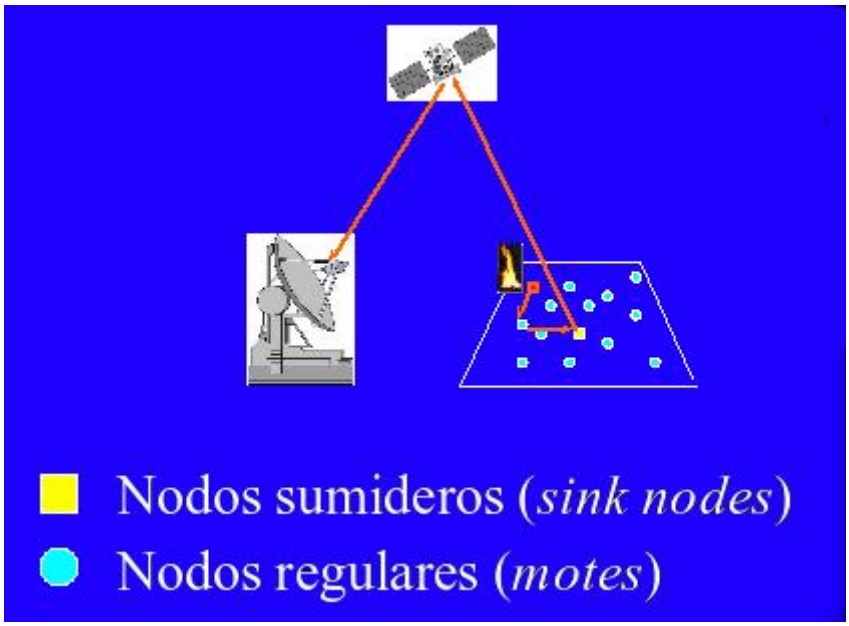

Figura 2: Posible arquitectura de una WSN para detección de incendios

Por supuesto, los dispositivos para este tipo de redes deben tener un **reducido consumo de energía** (por ser los sensores dispositivos electrónicos, van alimentados con una fuente limitada de energía -pilas convencionales, generalmente- y el consumo de ésta se presenta como uno de los principales problemas de las *WSN's*). Por otra parte, tienen una serie de limitaciones importantes como pequeña capacidad de cálculo, mínimo espacio de almacenamiento y una capacidad reducida de comunicación. Casi todas estas limitaciones vienen impuestas, principalmente, por un motivo: **suministro limitado de energía**.

El consumo energético es uno de los factores clave de las *WSN's* ya que, si se quieren desplegar redes del orden de los cientos o miles de nodos, no es factible sustituir las baterías de éstos una vez que termine su carga. Debido a esto, la vida útil de las baterías se convierte, automáticamente, en la vida útil del dispositivo. Por este motivo es conveniente desarrollar mejoras en el uso de la energía que hacen los dispositivos para permitir un ahorro energético y por consiguiente, alargar la vida útil de los aparatos. La transmisión de un solo bit equivale en gasto energético a la ejecución de cientos de instrucciones en el procesador. En capítulos posteriores, se tratarán, más a fondo, los protocolos más usados en la actualidad.

Capítulo 2: Los dispositivos

Las redes de sensores inalámbricos cuentan con varios fabricantes de nodos sensores:

- CROSSBOW: especializada en los sensores y que desarrolla plataformas hardware y software que dan soluciones para las WSN's. Entre sus productos, encontramos las plataformas MICA, MICA2, MICAZ, MICA2DOT, TELOS y TELOSB.

- MOTEIV: Joseph Polastre desarrolló Tmote Sky y Tmote Invent.

- SHOCKFISH: empresa suiza que desarrolló TinyNode.

Figura 3: Logotipos de los fabricantes

Los nodos sensores o motes más típicamente utilizados son los MICA2, diseñados por la Universidad de Berkeley y comercializados por Crossbow. Han sido los MPR410 de la serie MICA2 los empleados en nuestro proyecto. El fabricante dispone de diversos modelos con distintos tamaños y usos. En la figura se muestran los dos modelos más usados en la actualidad:

Figura 4: MICA2 y MICA2DOT

2.1 Hardware de los dispositivos

2.1.1 Baterías

El MPR400 (916MHz), MPR410(433MHz) y MPR420(315MHz) de la serie MICA2 son la última generación de “motas” de la empresa Crossbow. Esta serie está, por defecto, alimentada con dos baterías AA de 1'5V aunque, según las especificaciones del fabricante, los sensores pueden funcionar con cualquier fuente de alimentación que proporcione una tensión continua entre 2'7 y 3'3 voltios. Es posible utilizar cualquier combinación de baterías (AAA, C, D) o fuente de alimentación siempre que se encuentre dentro del rango especificado. En la tabla 1, se pueden observar las especificaciones extraídas del *datasheet* de los dispositivos.

2.1.2 Consideraciones de la antena

La interfaz radio de los MICA2 es capaz de realizar operaciones en múltiples canales dentro del ancho de banda establecido. El MPR420 puede trabajar en 4 canales de la banda de 315 MHz, el MPR410 puede actuar también en 4 canales pero de la banda de 433 MHz, y el MPR400 puede operar en 50 canales en una banda desde los 902 MHz hasta los 928 MHz. El fabricante especifica que, para que no exista interferencia, es necesario que los canales adyacentes estén separados un mínimo de 500 KHz. Tanto la serie MPR400(MICA2) como la MPR500(MICA2DOT) utilizan una interfaz radio compatible entre sí que permite la comunicación con nodos de la otra serie.

La potencia de transmisión en los MICA2 puede ser ajustada en un gran rango de niveles. Existe un registro que controla el nivel de potencia RF dentro del propio sensor (PA_POW, dirección 0x0B) y la tabla 2 proporciona los valores correspondientes a las distintas potencias para el MPR400.

La radio de los MICA2 también proporciona una medida de la potencia de señal recibida llamada RSSI. Esta salida es medida en el canal 0 del conversor A/D y puede ser leída desde los programas que se ejecutan. Esta medida será utilizada (como se mostrará más adelante) en este proyecto para calcular la distancia entre sensores y reducir, con ello, la potencia de transmisión, con el consiguiente ahorro de batería. La conversión desde el canal 0 del ADC al contador RSSI en dBm, viene dada por la siguiente expresión:

$$V_{RSSI} = \frac{V_{BATT} \cdot ADCCounts}{1024}$$

$$RSSI(dBm) = -51.3 \cdot V_{RSSI} - 49.2[dBm] \quad @ \quad 433\text{MHz}$$

Especificaciones del Sistema

Currents		Example Duty Cycle
Processor		
current (full operation)	8mA	1
current sleep	8uA	99
Radio		
current in receive	8mA	0.75
current transmit	12mA	0.25
current sleep	2uA	99
Logger Memory		
write	15mA	0
read	4mA	0
sleep	2uA	100
Sensor Board		
current (full operation)	5mA	1
current sleep	5uA	99
Computed mA-hr used each Hour		
Processor		0.0879
Radio		0.0920
Logger Memory		0.0020
Sensor Board		0.0550
Total current (mA-hr) used		**0.2369**
Computed battery life Vs. battery size		
		Battery Life (months)
Battery Capacity (mA-hr)		
	250	1.45
	1000	5.78
	3000	17.35

Tabla 1: Especificaciones de funcionamiento para la serie MICA2

La antena utilizada varía en función de los sensores para transmitir no viene incluida con los sensores, por tanto, es necesario estudiar bien el tipo de antena antes de colocarla. El alcance y la calidad de la transmisión van muy ligados al tipo de antena y a su posición dentro del entorno. Además, se debe estar seguro de que se cumple con las regulaciones internacionales en cuanto a radiación se refiere. En Crossbow, se ha usado una antena de cable de cobre sólido y considerando la longitud de la antena, existe una regla empírica: 75/frecuencia (MHz) = longitud en metros. Para los 433 Mhz de un MICA2, sería necesaria una antena de cable de cobre sólido de 17.3 cm. Si se desea un incremento del rango y el tamaño es una preocupación secundaria, las antenas direccionales pueden ser atractivas.

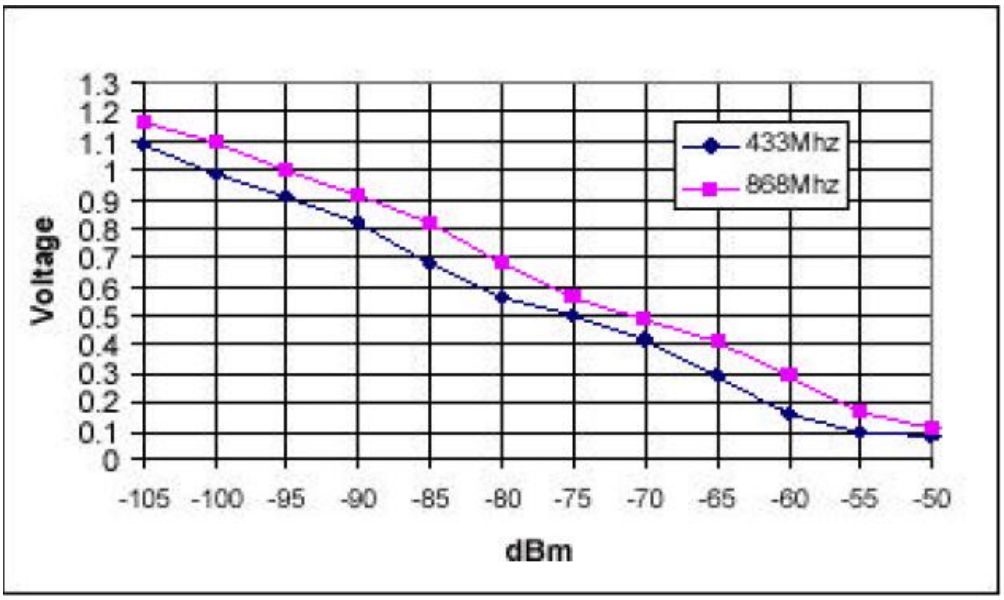

Figura.5: Representación de la tensión respecto a la potencia de transmisión

2.1.3 Almacenamiento de datos

Las “motas” MICA2 incluyen una memoria FLASH de 4 Mbit para almacenar datos, mediciones o cualquier tipo de información. La memoria FLASH soporta alrededor de 100.000 medidas.

2.1.4 Sensor Boards

Las series MTS de las placas de sensores y las series MDA de las placas de sensores y adquisición de datos son diseñadas para interactuar con la familia de Motes

inalámbricos MICA, MICA2 y MICA2DOT. Hay una variedad de placas de sensores disponibles y son específicas al tipo del nodo sensor. Dos placas muy simples son las siguientes:

- Basic Sensorboard:

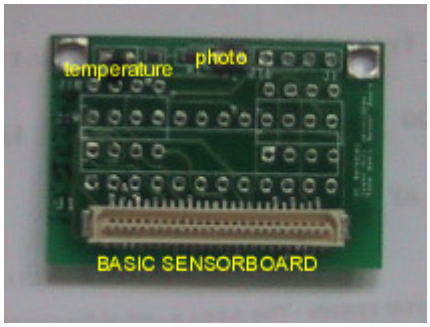

Figura.6: Basic Sensorboard

Esta placa tiene dos sensores:

- temperatura
- luz

Trabaja con las plataformas rene, rene2 y mica. Es la placa, por defecto, par alas plataformas rene y rene2.

- Mica Sensorboard:

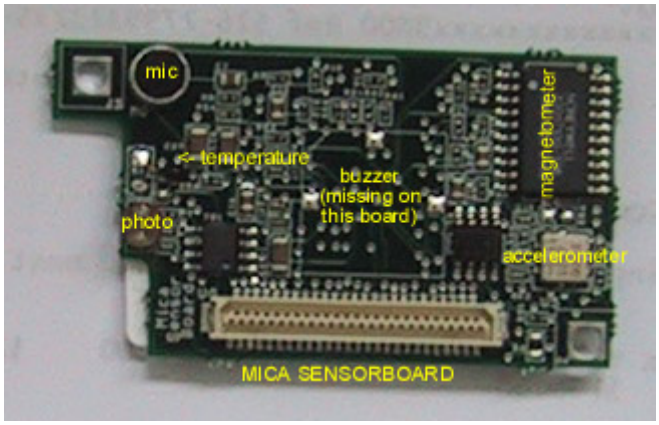

Figura 7: Mica Sensorboard

Esta placa puede tener los siguientes sensores:

- temperatura
- luz
- magnetómetro
- acelerómetro
- micrófono
- sonador (timbre)

Algunas placas no tienen todos los componentes arriba indicados. Es la placa, por defecto, para la plataforma mica.

2.1.5 Conector de Ampliación

Los Mica2 soportan una gran variedad de sensores que pueden ser añadidos mediante su “conector de expansión”. Este conector provee una interfaz que incluye entradas de un conversor analógico/digital (ADC) para leer la salida de los sensores, alimentación y tierra, control de potencia, interfaz *I2C*, interfaz *UART*, entrada/salida digital de propósito general...

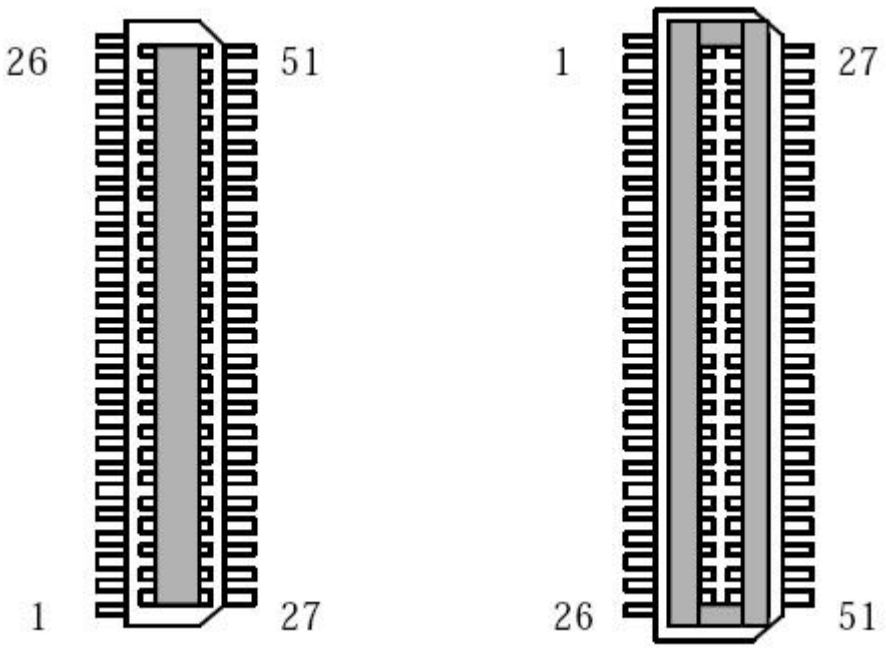

Figura 8: Conectores (macho y hembra) de expansión del MICA2

PIN	DESCRIPTION	PIN	DESCRIPTION
1	GND	27	UART_RXD0
2	VSNSR	28	UART_TXD0
3	INT3	29	PW0
4	INT2	30	PW1
5	INT1	31	PW2
6	INT0	32	PW3
7	BAT_MON	33	PW4
8	LED3	34	PW5
9	LED2	35	PW6
10	LED1	36	ADC7
11	RD	37	ADC6
12	WR	38	ADC5
13	ALE	39	ADC4
14	PW7	40	ADC3
15	USART1_CLK	41	ADC2
16	PROG_MOSI	42	ADC1
17	PROG_MISO	43	ADC0
18	SPI_CLK	44	THERM_PWR
19	USART1_RXD	45	THRU1
20	USART1_TXD	46	THRU2
21	I2C_CLK	47	THRU3
22	I2C_DATA	48	RESETN
23	PWM0	49	PWM1B
24	PWM1A	50	VCC
25	AC+	51	GND
26	AC-		

Tabla 2: Relación de los PIN y sus funciones en el conector de expansión del MICA2

2.2 Plataforma de programación

2.2.1 Introducción

El fabricante proporciona, dentro de la familia de productos MICA, unas placas de programación para los sensores que valen tanto para los MICA2 como para los MICA2DOT. La usada en este proyecto es la MIB510 que se muestra a continuación:

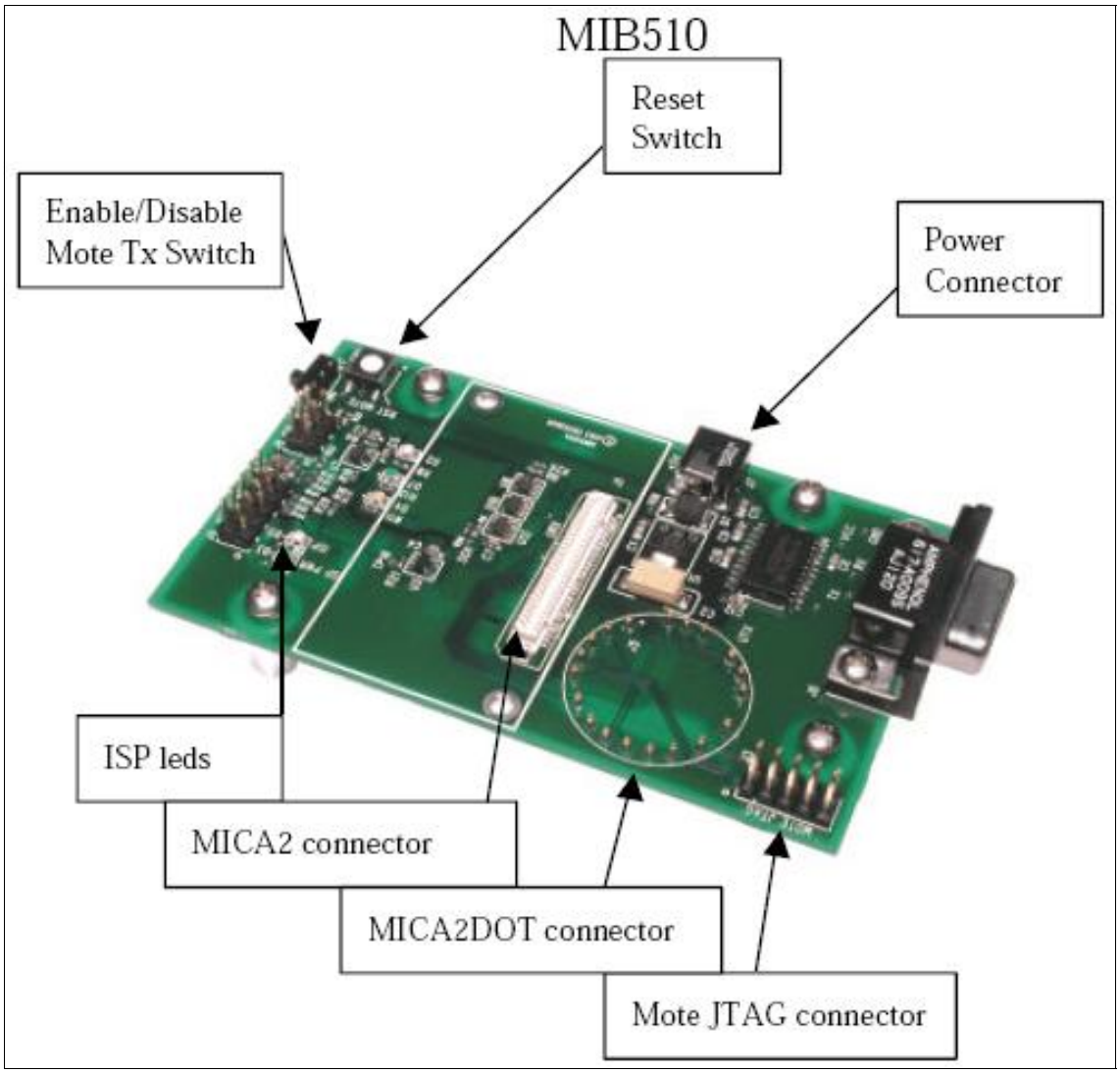

Figura 9: Placa de programación para los sensores MIB510

2.2.2 Programación de los dispositivos

Las placas de programación proporcionan alimentación a los sensores a través de un adaptador de corriente externo. También proporciona una interfaz serie (RS232) con un conector DB9 para la programación a través de un PC. Es importante, a la hora de programar, apagar el interruptor tanto de los MICA como de la placa, así como evitar que ésta última esté conectada a la corriente.

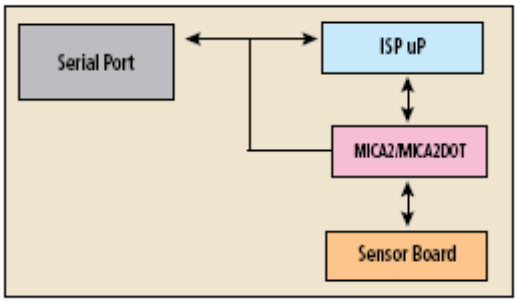

Figura 10: Diagrama de bloques de la placa MIB510

2.2.3 ISP

La placa de programación tiene un Procesador de Sistema Integrado (ISP) para programar los sensores. El código es descargado al ISP, a través del puerto serie, y el ISP programa el código en el sensor. El ISP y la “mota” comparten el puerto serie. El ISP trabaja a una velocidad fija de 115.000 baudios; continuamente, monitoriza los paquetes que entran por el puerto serie en busca de un patrón determinado de bytes, cuando lo detecta, desactiva la interfaz serie del sensor y toma el control del puerto serie.

2.2.4 Botón de Reset

La placa dispone de un botón de reset que, al ser presionado, resetea tanto el procesador del sensor como el ISP. El pulsador resetea primero el ISP, una vez reseteado, el ISP activa el pin de reset del procesador del sensor.

2.2.5 Alimentación de la placa

La MIB510 tiene un regulador de tensión que soporta entre 5V y 7V de tensión continua y proporciona a los dispositivos una tensión constante de 3V. Es importante tener cuidado a la hora de programar los MICA2 ya que la alimentación en este caso debe estar desconectada así como el interruptor de batería de los MICA, que también debe estar desconectado.

2.3 Lenguajes de programación para los dispositivos

2.3.1 Introducción

La programación de sensores es complicada, entre otras dificultades está la limitada capacidad de cálculo y la cantidad de recursos. Al igual que en los sistemas informáticos tradicionales, aquí encontramos entornos de programación prácticos y eficientes para depurar código, simular…

Podemos encontrar lenguajes como:

nesC	Lenguaje que utilizamos para nuestras motas, y que está directamente relacionado con TinyOS.
Protothreads	Específicamente diseñado para la programación concurrente, provee hilos de dos bytes como base de funcionamiento
SNACK	Facilita el diseño de componentes para redes de sensores inalámbricas, sobre todo cuando la información o cálculo a manejar es muy voluminoso. Comparando con nesC, este lenguaje hace su programación más fácil y eficiente.
c@t	Computation at a point in space (@) Time.
DCL	Lenguaje de composición distribuido (Distributed Compositional Language).
galsC	Diseñado para ser usado en TinyGALS, es un lenguaje programado mediante el modelo orientado a tarea, fácil de depurar, permite concurrencia y es compatible con los módulos nesc de TinyOS.
SQTL	(Sensor Query and Tasking Language): es una interesante herramienta para realizar consultas sobre redes de motas.

Tabla 3: Lenguajes de programación para motas

2.3.2 nesC

2.3.2.1 Introducción

NesC es un lenguaje de programación que es empleado en la creación de aplicaciones. Dichas aplicaciones serán ejecutadas en nodos sensores que contengan el Sistema Operativo TinyOS. El hecho que hace que nesC sea un lenguaje cómodo es que se base en una *programación orientada a componentes*. Así, se consigue una filosofía que permita abstraer al programador de bastantes detalles de bajo implementación presentes en el Sistema Operativo.

La idea que hay detrás de este tipo de programación es que el propio Sistema Operativo, junto a los fabricantes de los nodos sensores, proporcione de forma intrínseca ciertos componentes ya implementados que ofrecen al programador una gran cantidad de funciones y utilidades para que el programador se centre en programar sólo la funcionalidad que desea en el dispositivo sin necesidad de tener que preocuparse por otros aspectos secundarios.

Por otro lado, nesC combina ciertos aspectos de la orientación a objetos, en el sentido de que se basa en una *programación orientada a interfaces y a eventos*, de manera que posee un manejador de eventos propio de este tipo de lenguajes, al igual que ocurre en Visual Basic.

Todos los componentes que ofrece, de forma intrínseca, el Sistema Operativo se van a denominar, a partir de ahora, componentes primitivos y los componentes proporcionados por terceros se van a denominar componentes complejos.

Cuando se dice que es un lenguaje orientado a objetos, se quiere decir que todos los componentes, tanto primitivos como complejos, proporcionan unas interfaces y si un programador desea utilizar un componente, la forma de hacerlo es usando dichas interfaces, pero recordemos que son interfaces en el sentido de Orientación a Objetos, por lo que en un momento dado se podría cambiar un componente por otro siempre y cuando este otro proporcionase la misma interfaz y dicho cambio no afectase al código de la implementación de la aplicación. De esta explicación se puede entresacar que se van a tener dos partes diferenciadas, como mínimo, en cada componente:

- Una parte de implementación que estará programada hacia componentes y...
- Una parte de configuración que permitirá decidir que componentes son los que se utilizan para proporcionar dichas interfaces a mi componente, lo cual se denomina wiring.

Como la forma de programar no es del todo secuencial, se recurre a la orientación a eventos, que atiende a una programación basada en eventos, de manera que se programan las acciones que se desean realizar cuando se produzca un evento.

A modo de resumen, se reflejan, a continuación, varias pinceladas de lo que modelo de programación nesC ofrece :

1) El modelo de NesC está formado por interfaces y componentes.

2) Una interfaz puede ser usada o puede ser provista, las componentes son módulos o configuraciones.

3) Una aplicación se verá representada como un conjunto de componentes, agrupados y relacionados entre sí, tal como se observa en la figura 9.

4) Las interfaces se utilizan para operaciones que describen la interacción bidireccional; el proveedor de la interfaz debe implementar comandos, mientras que el usuario de la interfaz debe implementar eventos.

5) Existen dos tipos de componentes:

- Módulos, que implementan especificaciones de una componente.
- Configuraciones, que se encargarán de unir (*wire*) diferentes componentes en función de sus interfaces, ya sean comandos o eventos.

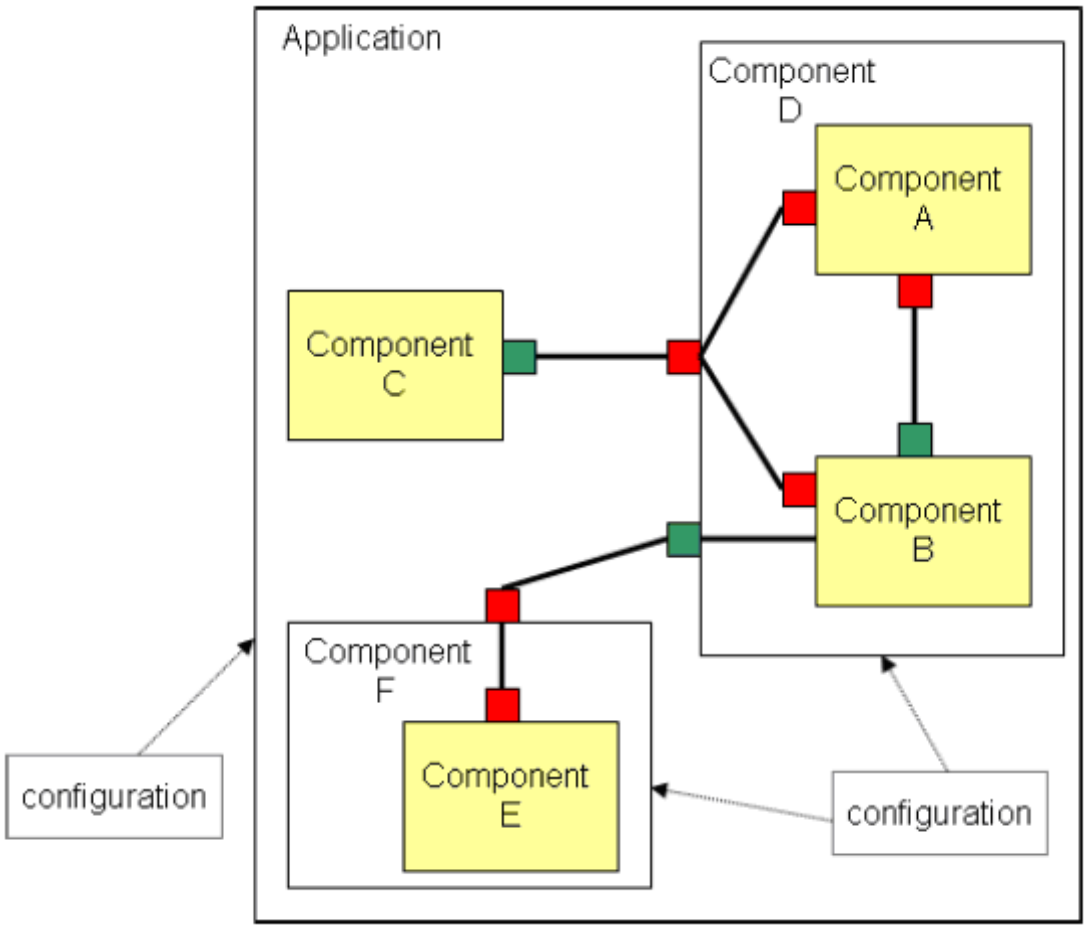

Figura 11: Aplicación genérica

2.3.2.2 Estructura de un componente

Un componente está formado, lógicamente, por varias secciones: *Configuration*, *Implementation* y *Module*. Deben estar obligatoriamente presentes en cualquier componente, aunque pueden estar vacías.

El estándar de TinyOS determina que las secciones de *Configuration* e *Implementation* han de ir en un fichero que recibirá el nombre del componente con la extensión *.nc* y la sección *Module* deberá ir en otro fichero que recibirá el nombre del componente concatenado con 'M' (M de *Module*), teniendo también la extensión *.nc*.

Una buena costumbre consiste en crear un fichero header o cabecera con extensión .h que contenga todas las enumeraciones, registros o tipos de datos creados por el usuario de los que hace uso la aplicación. La forma de ligar dicho fichero con los otros dos es utilizando al principio de los otros ficheros la directiva includes header; aunque como mención especial decir que si nos fijamos mejor en este directiva se puede ver que no se incorpora la extensión .h en la misma.

- **Configuración**: sirve para la configuración del componente (utilizado generalmente para crear librerías). Por lo general, no va a contener nada pero tiene que estar presente.

- **Implementación**: comúnmente, se le denomina “wiring” y no se corresponde a lo que nosotros pensamos por implementación. Si la implementación de nuestra aplicación utiliza interfaces, éstas han de ser proporcionadas por otro componente.

- **Módulo**: se corresponde a lo que nosotros pensamos por implementación. Está escrito en nesC e incluye todas las tareas que queremos que sean ejecutadas. La estructura del módulo será de la siguiente manera:

```
Module PracticaM {
 Provides
 Uses
 Implementation
 }
```

A continuación, se explicará cada una de las partes de la sección *Module*:

1) Provides: especifica las interfaces que va a proporcionar nuestro componente. Para proporcionarla, tendremos que tener implementadas las funciones de dicha interfaz.

2) Uses: especifica las interfaces que usa nuestro componente. Pero no sabe qué componentes proporcionan dichas interfaces ya que esto se establece en el Wiring.

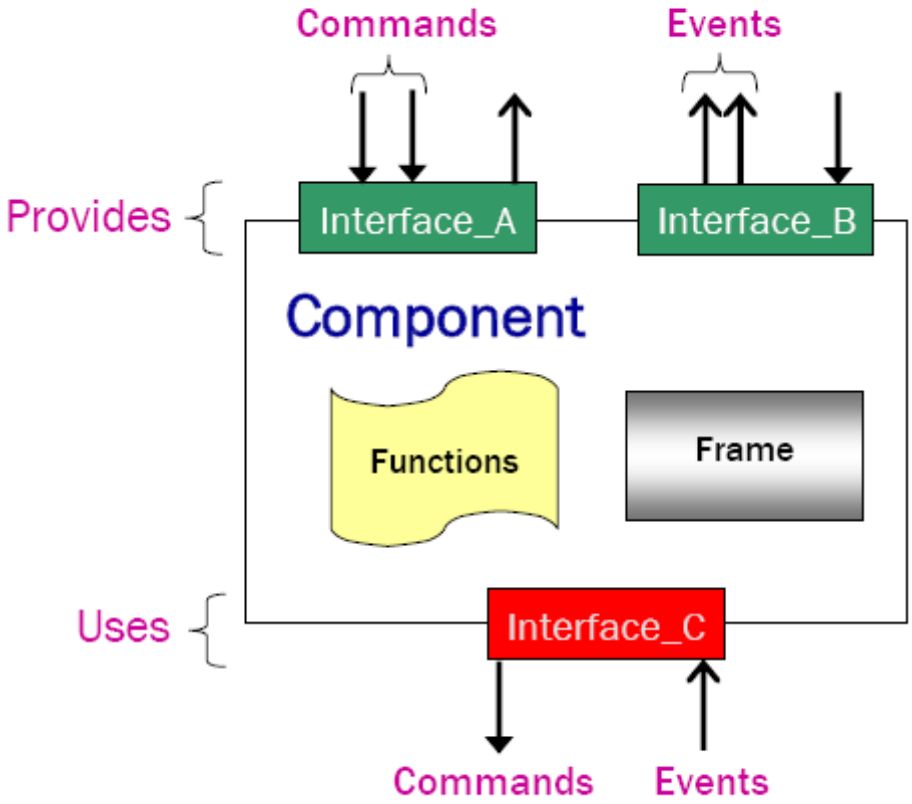

Figura 12: Componente genérico

Cuando usemos una interfaz:

- Podremos llamar a sus métodos.
- Tendremos que implementar los eventos que puedan producirse.
- Habrá que realizar el Wiring en el fichero correspondiente para indicar qué componente proporciona dicha interfaz.

3) Implementation: en esta parte es en donde se programa el comportamiento de nuestra aplicación. Deberá poseer las variables globales, las funciones de las interfaces que proporciono y los eventos de las interfaces que utilizo.

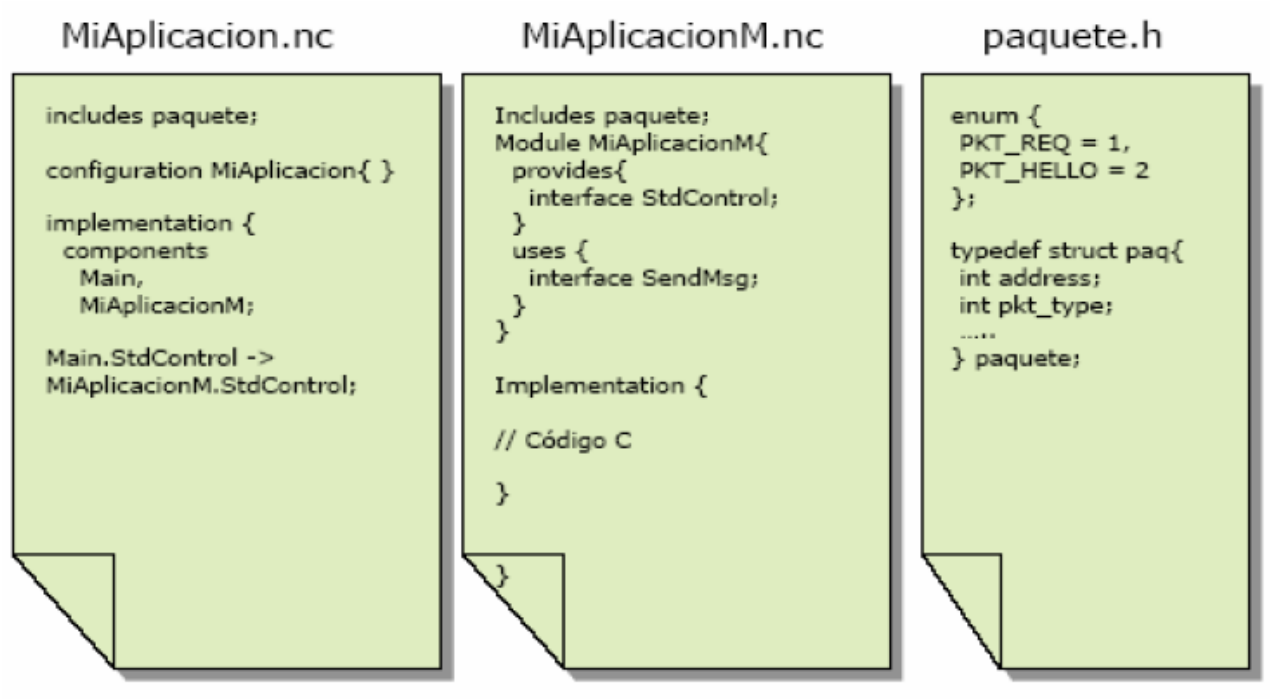

Figura 13: Ejemplo de un componente

2.3.2.3 Tipos de datos

Los tipos que pueden utilizarse son todos aquellos que proporciona C estándar más otros específicos de nesC, los cuales no aportan potencia de cálculo, pero son muy útiles para la construcción de paquetes ya que proporcionan al usuario información acerca del número de bits que ocupan, lo cual es muy importante a la hora de transmitir información vía radio.

Los tipos adicionales son los siguientes:

uint16_t	entero sin signo de 16 bits
uint8_t	entero sin signo de 8 bits
bool	es un boolean (TRUE, FALSE)
result_t	es un boolean pero (SUCCES, FAIL)

Tabla 4: Tipos de datos no estándares de C

En nesC, es posible la utilización de memoria dinámica, lo cual no es muy recomendable a no ser que sea absolutamente necesaria. Para utilizar la memoria dinámica, existe un componente especial denominado *MemAlloc.*

Como veremos en el apartado “2.3.2.4 Tipos de funciones”, existen distintos tipos de funciones o métodos, uno de ellos son las funciones asíncronas, las cuales no se realizan de forma inmediata y, por tanto, cuando usan una variable global, tienen que indicarlo de forma explícita para poder realizar una exclusión mutua de la memoria y no perder ningún valor. La forma de indicarlo es anteponiendo la palabra reservada “norace” delante de la declaración de la variable.

2.3.2.4 Tipos de funciones

Como el origen de nesC es el lenguaje de programación C, nesC hereda unas funciones clásicas que tienen la misma semántica que en C y con la misma forma de invocarlas.

Pero existen otros tipos de funciones en nesC: *task, event y command.*

Las funciones “**command**” o comandos son funciones como las clásicas, pero que se ejecutan de forma síncrona (cuando se llaman, se ejecutan inmediatamente). La forma de llamar estas funciones es: *call interfaz.nombreFuncion*

Las funciones “**task**” o tareas se ejecutan concurrentemente en la aplicación, al igual que ocurre con los hilos o threads. Se invoca con lo siguiente: post interfaz.nombreTarea e inmediatamente, tras su invocación, continúa la ejecución del programa invocador.

Las funciones “**event**” o eventos son llamadas cuando se levanta una señal en el sistema. Poseen la misma filosofía que la programación orientada a eventos, de manera que cuando el componente recibe un evento, se realizará la invocación de dicha función. Aunque existe un método para poder invocar manualmente este tipo de funciones y es el siguiente: *signal interfaz.nombreEvento*, además en el caso de tratarse de interfaces parametrizadas, se podrá utilizar la variante *signal interfaz.nombreEvento[parámetro]*.

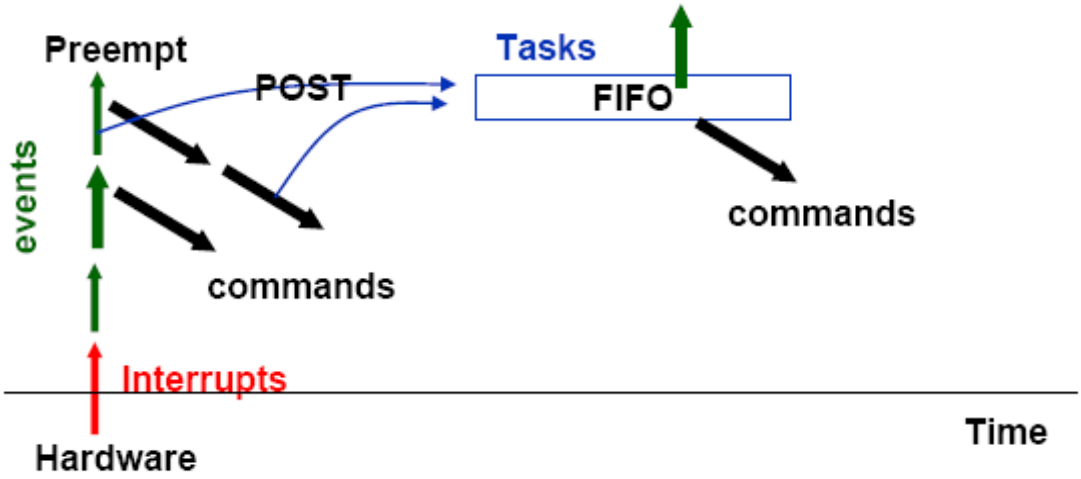

Figura 14: Esquema de la ejecución de una aplicación en nesC

En la declaración de estas tres funciones se puede anteponer la palabra reservada *async* para indicar que poseen una ejecución asíncrona. Este tipo de ejecución, por lo general, viene dado cuando la ejecución de uno de estos tipos de función viene determinada por el levantamiento de una señal hardware. Por ejemplo, que la temperatura ya se encuentre preparada para ser leída. Para llamar a una de estas funciones asíncronas, si se intenta acceder a una variable global, que habrá tenido que ser declarada con *norace* para indicar su exclusión mutua.

2.4 Sistemas Operativos para los dispositivos

2.4.1 Introducción

Existe una gran variedad de sistemas operativos para motas. Están presentados en la siguiente tabla.

Bertha (pushpin computing platform)	Se trata de una plataforma de software diseñada e implementada para modelar, testear y desplegar una red de sensores distribuida de muchos nodos idénticos. Sus principales funciones se dividen en los siguientes subsistemas: - Administración de procesos - Manejo las estructuras de datos - Organización de los vecinos - Interfaz de Red
Nut/OS	Es un pequeño sistema operativo para aplicaciones en tiempo real, que trabaja con CPU's de 8 bits. Tiene las siguientes funciones: - Multihilo - Mecanismos de sincronización - Administración de memoria dinámica - Temporizadores asíncronos - Puertos serie de Entrada/Salida Está diseñado para procesadores con los siguientes recursos: - 0.5 kBytes RAM - 8 kBytes ROM - Velocidad de 1 MIPS CPU
Contiki	Es un Sistema Operativo de libre distribución para usar en un limitado tipo de computadoras, desde los 8 bits a sistemas embebidos en microcontroladores, incluidas motas de redes inalámbricas.

CORMOS (Communication Oriented Runtime System for Sensor Networks)	Es específico para redes de sensores inalámbricas, como su nombre indica.
Ecos (embedded Configurable operating system)	Es un sistema operativo gratuito, en tiempo real y diseñado para aplicaciones y sistemas embebidos que sólo necesitan un proceso. Se pueden configurar muchas opciones y puede ser personalizado para cumplir cualquier requisito, ofreciendo la mejor ejecución en tiempo real y minimizando las necesidades de hardware.
EYESOS	Se define como un entorno para escritorio basado en Web, permite monitorizar y acceder a un sistema remoto mediante un sencillo buscador.
MagnetOS	Es un sistema operativo cuyo objetivo es ejecutar aplicaciones de red que requieran bajo consumo de energía, adaptativas y fáciles de implementar.
MANTIS (MultimodAl NeTworks In-situ Sensors)	
TinyOS	Sistema Operativo utilizado para MICA2, el cual se explicará minuciosamente en la Sección 2.3.1.
t-Kernel	Es un sistema operativo que acepta las aplicaciones como imágenes de ejecutables en instrucciones básicas. Por ello, no importará si está escrito en C++ o lenguaje ensamblador.
LiteOS	Sistema operativo desarrollado en principio para calculadoras, pero que ha sido también utilizado para redes de sensores.

Tabla 5: Sistemas Operativos para nodos sensores

2.4.2 TinyOS

2.4.2.1 ¿Qué es TinyOS?

TinyOS es un sistema operativo open source para redes de sensores inalámbricas y que se basa en componentes. Este sistema, junto a sus librerías y aplicaciones, está escrito en el lenguaje de programación nesC como un conjunto de tareas y procesos que colaboran entre sí y es útil para pequeños dispositivos, tales como las motas. Es un sistema operativo "event-driven", lo que quiere decir que funciona a partir de eventos producidos que llamarán a funciones. Está diseñado para incorporar nuevas innovaciones rápidamente y para funcionar bajo las importantes restricciones de memoria que se dan en las redes de sensores. El entorno de desarrollo de TinyOS soporta directamente la programación de diferentes microprocesadores y permite programar cada tipo con un único identificador para diferenciarlo, o lo que es lo mismo se puede compilar en diferentes plataformas cambiando el atributo. TinyOS está desarrollado por un consorcio liderado por la Universidad de California en Berkeley en cooperación con Intel Research.

En nesC, los programas están compuestos por componentes que se enlazan para formar un programa completo. Los componentes se enlazan a través de sus interfaces. Estas interfaces son bidireccionales y especifican un conjunto de funciones que están implementadas bien por los proveedores o bien por los que la utilizan. NesC esperará que el código que va a ser generado cree un programa con un ejecutable que contenga todos los elementos del mismo, así como los manejadores de las interrupciones de programas de más alto nivel.

En la figura siguiente, se muestra un diagrama de bloques en el que se describe el proceso de compilación para una aplicación en TinyOS.

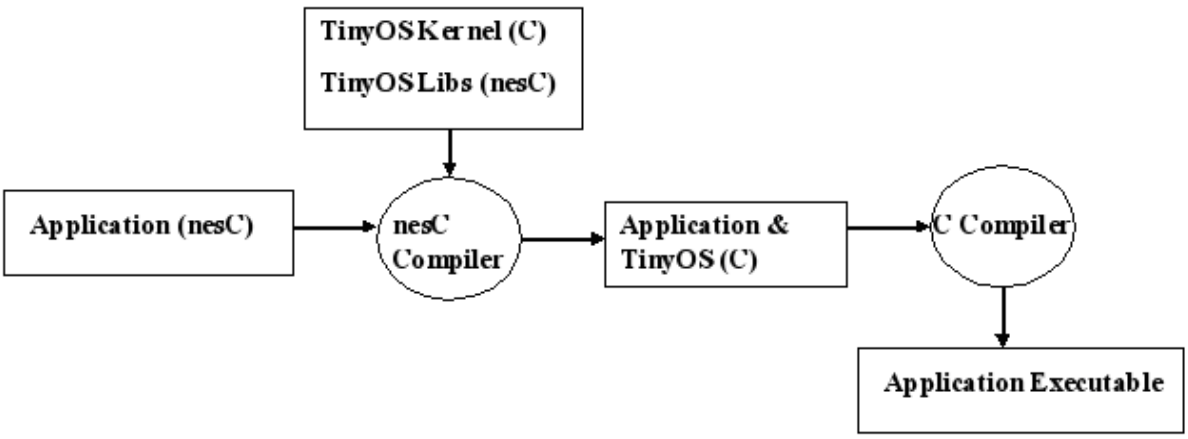

Figura 15: Compilación de una aplicación TinyOS

TinyOS es un proyecto de la Universidad de Berkeley, es por esto que el foco principal de información se encuentra en la página del proyecto presente en el servidor de dicha Universidad. A continuación se mencionan algunos links importantes.

Dirección	Contenido
http://www.tinyos.net	Página Principal del Proyecto.
http://www.tinyos.net/download.html	Sección donde se pueden bajar las ultimas versiones del sistema operativo
http://www.tinyos.net/support.html	Página con el soporte para TinyOS
http://www.tinyos.net/tinyos-1.x/doc/tutorial/index.html	Tutorial on-line.
http://sourceforge.net/projects/tinyos/	Control de Versiones (CVS) de TinyOS, en el que se pueden encontrar tanto aplicaciones, que se encuentran en desarrollo para implementarse con TinyOS, como el desarrollo del código fuente de éste.

Tabla 6: Principales links de TinyOS

2.4.2.1 Componentes primitivos de TinyOS

Los tipos de componentes que podemos encontrar en TinyOS pueden ser primitivos o compuestos (proporcionados por una librería o una aplicación).

Los componentes primitivos principales en TinyOS son:

1) Componente Main:

Representa el cuerpo main al estilo de C y necesita de la interfaz StdControl para su funcionamiento. Dicha interfaz será proporcionada por el componente que quiera convertirse en una aplicación.

- **Interfaz StdControl**: es una interfaz especial que deben proporcionar todos los componentes que se quieran llamar aplicación y, por tanto, sean ejecutables en un sensor. Dicha interfaz obliga a implementar los siguientes métodos:

command result_t init();

Se ejecutará cuando arranque la mota. Se inicializarán las variables globales, etc. y se llamarán a los métodos init() de los componentes que se utilizan (no todos, sino los que sean necesarios).

command result_t start();

Se ejecutará después del método init() y cuando la mota pase de off a on. Se arrancarán los temporizadores y se llamarán a los métodos start() de los componentes que se utilizan (sólo los necesarios).

command result_t stop();

Se ejecutará cuando se apague la mota o se suspenda (estado idle). Se harán llamadas a los métodos stop que utilizo y en los que sea necesario.

***2) Componentes GenericComm y GenericPromiscuousComm*:**

Este componente sirve para enviar y recibir paquetes vía radio y vía puerto serie (UART). Actúa como switch entre la radio y el puerto serie, de manera que si un paquete se envía vía radio a la dirección del puerto serie, cuando lo reciba este componente, será enviado dicho paquete por el puerto serie.

Utiliza el modelo AM estándar de TinyOs por lo que se basa en el envío de paquetes TOSMsg, los cuales poseen la siguiente estructura:

uint16_t addr;	***uint8_t type;***	***uint8_t group;***	***uint8_t length;***	***int8_tdata [TOSH_DATA_LENGTH];***	***uint16_t crc;***

El significado de cada uno de los campos es el siguiente:

addr	Es la dirección a la que va destinado el paquete, existen ciertas direcciones especiales: o *TOS_BCAST_ADDR* – Es la dirección de broadcast o *TOS_LOCAL_ADDRESS* – Es la dirección local (localhost) o *TOS_UART_ADDR* – Es la dirección del puerto serie
type	Es el tipo de paquete que se está enviando, este tipo se utiliza en las interfaces paramétricas para determinar qué instancia de la interfaz se va a hacer cargo de procesar dicho paquete.
group	Es el grupo al que pertenece el sensor que está enviando el paquete, de manera que sólo lo reciban aquellos sensores que pertenezcan al mismo grupo. De esta manera, puede coexistir más de una red de sensores distinta sobre el mismo medio físico aéreo y sobre el mismo canal de radio.
length	Es la longitud del paquete.
CRC	Corrección de errores.
data	Son los datos que se enviarán, los cuales pueden formar una estructura que conforme un nuevo tipo de paquete de nivel superior.

Tabla 7: Campos de un paquete TOSMsg

Estos componentes proporcionan las siguientes interfaces:

SendMsg [TIPO_PAQUETE]	Esta interfaz posibilita el poder enviar paquetes a un determinado destino, estos paquetes han de ser del tipo que se especifique en el parámetro de la interfaz. Proporciona mecanismos para poder enviar mensajes a otros sensores o al puerto serie del ordenador. Esta interfaz obliga a implementar el siguiente método: *command result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg);* que sirve para enviar el mensaje a la dirección *address.* El mensaje es una variable TOS_MsgPtr, que se trata de un puntero a una estructura TOS_Msg y que deberá ser declarada como global. Si se declarase como local, la duración de ese mensaje sería tan corta que no se habría enviado el mensaje antes de terminar de ejecutarse la función.

	Además, el componente que utilice esta interfaz deberá implementar el siguiente evento: event result_t sendDone(TOS_MsgPtr msg, result_t success); el cual se producirá cuando el envío de un mensaje haya sido satisfactorio.
ReceiveMsg [TIPO_PAQUETE]	Esta interfaz posibilita el poder recibir paquetes de un tipo determinado, especificado en el parámetro de la interfaz. Un sensor recibirá un paquete siempre y cuando el paquete sea enviado por un sensor perteneciente al mismo grupo y que el destino sea de broadcast o coincida con nuestra dirección local. En el último caso, si se trata del componente ***GenericCommPromisuos*****,** se recibirá siempre aunque no sea broadcast, ni vaya dirigido hacia nosotros. No es necesario implementar ningún método pero sí el siguiente evento: *event TOS_MsgPtr receive(TOS_MsgPtr m);* el cual se producirá ante la llegada de un paquete. Dicho evento devolverá, al final de su invocación, el mismo paquete que se ha recibido, de manera que se pueda pasar a capas de nivel superior.
StdControl	No hay que olvidar que si un componente proporciona la interfaz StdControl, es porque se trata de una aplicación y, por tanto, es necesario llamar a los métodos de dicha interfaz a la vez que a los de nuestro componente.

Tabla 8: Interfaces de los componentes GenericComm y GenericPromiscuousComm

3) Componente ADCC:

Este componente tiene la misión de ofrecer mecanismos para poder recoger la información de los sensores que posea la placa. Será empleado, en nuestro proyecto, para tomar los datos de la fuerza de la señal de los paquetes recibidos.

Proporciona dos interfaces:

ADC [puerto]	Esta interfaz posibilita el poder realizar una conversión analógico/digital de un puerto. Cada uno de los sensores conectados a la placa tiene asignado un puerto, de manera que, mediante la especificación del parámetro de la interfaz, se decide cuál de los sensores es el que se quiere procesar para obtener su valor. Obliga a implementar el método: *async command result_t getData();*
ADC [puerto]	que iniciará un muestreo del valor que posee el sensor. Cuando dicho muestreo haya finalizado, se producirá el evento *dataReady().* También obliga a implementar el siguiente método: *async command result_t getContinuousData();* que es muy parecido al anterior, aunque no produce un único muestreo, sino una secuencia continua de ellos (es como tener un timer que invoque al método *getData()* cada cierto tiempo). Además de los métodos anteriores, deberá implementarse el siguiente evento: *async event result_t dataReady(uint16_t data);* el cual se producirá cuando se haya realizado un muestreo y el valor del mismo esté en la variable *data*. Se trata de un evento asíncrono, por lo cual, si deseo almacenar dicha variable, necesitaré declarar la variable donde voy a almacenar el valor con la palabra reservada *norace.*
ADCControl [puerto]	Esta interfaz posibilita realizar configuraciones acerca del muestreo de las muestras capturadas por el componente, de manera que se puede configurar su velocidad de muestreo o realizar un remaping de los puertos, para establecer una correspondencia entre los puertos software (comunes a todos los modelos de sensores) y los puertos hardware dependientes del tipo de placa sobre el que se vaya a ejecutar la aplicación.

Tabla 9: Interfaces del componente ADCC

2.4.2.2 Estructura de TinyOS

Los directorios que forman el Sistema Operativo tienen la siguiente estructura:

/tos/interfaces: Contiene todas las interfaces que son proporcionadas por los componentes primitivos y por las aplicaciones de ejemplo.

/tos/lib: Contiene librerías para resolver determinados problemas.

/tos/system: Contiene todos los componentes primitivos que proporciona TinyOs.

/tos/types: Contiene los tipos que se utilizan en las primitivas de TinyOs.

/tos/platform: Contiene los ficheros necesarios para la ejecución en las diversas plataformas.

/tos/sensorboard: Contiene los ficheros que son específicos de cada placa.

Tabla 10: Directorios de TinyOS

Capítulo 3: Protocolos MAC para Redes Inalámbricas de Sensores

3.1 Introducción

Un protocolo de MAC tiene que servir de base para protocolos de más alto nivel. En el caso de redes de sensores, se tendría el protocolo de enrutamiento, que usará las funciones implementadas en la MAC para enviar y recibir paquetes, sincronizar sus operaciones, etc.

Para entender cuáles son las necesidades ante la construcción de un protocolo a nivel de la capa MAC, se dan, a continuación, unas pinceladas de las características que deben poseer:

La **flexibilidad**, porque el entorno inalámbrico es totalmente cambiante debido a interferencias en el aire de otras ondas, propiedades y formas de los materiales del entorno, y un largo etcétera. Además, los nodos pueden fallar en cualquier momento, teniendo que buscar nuevos caminos, reconfigurando la red y recalibrando los parámetros. El tráfico también puede incrementarse, ya que la información requerida en cada momento es cambiante.

La **eficiencia**; un protocolo de MAC debe ser eficiente para poder trabajar en tiempo real, debe ser fiable y robusto ante las interferencias, tolerante a los ruidos... Además debe estar profundamente integrado con el medio donde va a trabajar, a su vez que debe ser un software “fácil de implementar” y con funciones que cubran las necesidades más amplias del mercado. Estos protocolos afectan directamente al consumo energético, ya que son la capa más próxima al nivel físico, también determinan, en parte, el coste del sistema. Así como son clave a la hora de especificar la latencia y el nivel de seguridad del sistema. En las redes de sensores, estos protocolos determinan los canales de radio a utilizar, implementan las transmisiones y recepciones a bajo nivel, además de controlar los errores. Las funciones de un protocolo MAC son, entre otras, controlar el acceso al medio compartido, que en este caso será un canal de radio (a través del aire). El protocolo debe evitar las interferencias entre transmisiones, mitigando el efecto de las colisiones, evitándolas en la medida de lo posible o detectándolas en el caso de que se den.

Los protocolos de control de acceso al medio (MAC) han sido investigados durante décadas; los diseños varían mucho según el objetivo de la aplicación. Pueden ser clasificados en categorías basadas en diferentes principios; algunos son centralizados, con una estación central como líder del grupo haciendo el control de acceso, otros son distribuidos. Algunos usan un único canal, otros varios. Algunos usan diferentes versiones de acceso aleatorio, otros usan reserva de canal y planificación. Los protocolos están optimizados para diferentes objetivos: energía, retardo, tasa de transferencia, equidad,

calidad de servicio (QoS) o soporte de múltiples servicios. Cada tipo de red necesita un protocolo diferente. Por ejemplo, las redes donde los eventos se producen de forma periódica necesitan protocolos que usen reserva y planificación del tiempo. Tendrán una mejor utilización del canal, y tendrán un mayor tiempo de vida. Por el contrario, para las redes de sensores con eventos asíncronos, el protocolo MAC ha de ser distribuido y optimizado para el ahorro de la energía. Un método distribuido usando múltiples canales y acceso aleatorio es lo más indicado para estas redes, se evita el tener un único punto de fallo, múltiples canales reducen las colisiones y retransmisiones, además del incremento de la tasa de transferencia. El acceso aleatorio evita al nodo conocer la red, lo que hace que no sea necesaria la sincronización.

3.2 Clasificación de Protocolos MAC

Existe una gran variedad de protocolos que fueron desarrollados con anterioridad para otros tipos de redes, y que, ahora, han sido implementados sobre redes inalámbricas de sensores. Además, diversos grupos de trabajo han creado otros protocolos ad hoc para este tipo específico de redes, lo que nos da una amplia gama de protocolos, que se enumeran a continuación según su tipo:

Protocolos de Acceso Múltiple por Detección de Portadora (CSMA. Carrier Sense Multiple Access) o ranurados
» Sensor MAC (S-MAC) » 802.15.4 » Timeout MAC (T-MAC) » Berkeley-MAC (B-MAC) » Power-Aware Multi-access protocol with signaling (PAMAS)
Protocolos de Acceso multiplexado por división de tiempo (TDMA. Time Division Multiplex Access)
» Traffic-Adaptive MAC (TRAMA) » Low-Energy Adaptive Clustering Heirarchy(LEACH) » Power Aware Clustered TDMA (PACT) » Bit-Map Assisted (BMA) » Proposed Gateway MAC (G-MAC) » SPRIME (SupSlot Period Reservation & Inter-Master Estimation) » SMACS (Self-Organized MAC for Sensor networks)
Otros
» DMAC (Dynamic Topology MAC) » CMAC (Spatial Correlation-based Collaborative) » DSMAC (Dynamic Duty Cycle for WSN) » SmartNode (*send with same power as received) » STEM (Sparse Topology and Energy Management) » DPSM (Dynamic Power Saving Mechanism) » MACA (MultiAccess Collision Avoidance)

» ZMAC (Hybrid MAC for WSN)

Tabla 11: Clasificación de protocolos a nivel MAC

3.3 Elección del protocolo MAC

La elección de un buen protocolo MAC, nos obliga a tener en cuenta los siguientes aspectos:

• Escalabilidad: porque las redes de sensores son, por definición, dinámicas y el añadir nodos es totalmente normal, luego debe estar preparado para trabajar con diferentes números de nodos.

• Debe ser posible el predecir los retrasos, implementando los protocolos un mecanismo que evite tener que preocuparse del correcto funcionamiento en función de la disposición de los nodos, proximidad, calidad del canal…

• Adaptabilidad a los cambios mencionados anteriormente.

• Eficiencia a la hora de gestionar la energía, como principal desafío de las redes de sensores. La cantidad de energía utilizada en el envío y recepción de paquetes en las redes inalámbricas es esencial, ya que a menor energía utilizada, mayor tiempo de vida para la red. Varios protocolos MAC han sido desarrollados para enfrentarse al agotamiento de las baterías. S-MAC, por ejemplo, divide la estructura de envío, recepción de mensaje en períodos de escucha y sueño (durante el sueño el sensor usa el mínimo de energía). El período de escucha esta dividido en un período de sincronización y otro de transferencia de datos. La sincronización permite a los nodos anunciar periódicamente su planificación de tiempos, corrigiendo así los desplazamientos temporales propios de un sistema impreciso. Además, permite a la red sincronizar los períodos de sueño de los nodos, formando así un conjunto virtual de nodos que activan al mismo tiempo los períodos de escucha y sueño. El T-MAC (Timeout MAC) es similar al S-MAC, pero con un período de escucha adaptativo basado en el tráfico de la red. Con T-MAC, los nodos pueden ir directamente a la fase de sueño tan pronto como el tráfico de la red se termine.

• Fiabilidad, evitando los bloqueos, la pérdida de paquetes, la desaparición de nodos y respondiendo a interferencias o ataques externos a la red.

En relación al penúltimo punto (gestión de la energía), varios protocolos MAC han sido desarrollados para enfrentarse al agotamiento de las baterías. S-MAC, por ejemplo, divide la estructura de envío, recepción de mensaje en períodos de escucha y sueño (durante el sueño el sensor tiene un consumo mínimo). El período de escucha esta dividido en un período de sincronización y otro de transferencia de datos. La sincronización permite a los nodos anunciar periódicamente su planificación de tiempos, corrigiendo así los desplazamientos temporales propios de un sistema impreciso.

Además, permite a la red sincronizar los períodos de sueño de los nodos, formando así un conjunto virtual de nodos que activan al mismo tiempo los períodos de escucha y sueño. El T-MAC (Timeout MAC) es similar al S-MAC, pero con un período de escucha adaptativo basado en el tráfico de la red. Con T-MAC, los nodos pueden ir directamente a la fase de sueño tan pronto como el tráfico de la red se termine. A continuación, se detallan los cinco puntos clave para desarrollar un protocolo MAC que presente un consumo mínimo en una red de sensores inalámbrica:

1. Las colisiones deben ser evitadas siempre que sea posible, ya que la retransmisión produce un innecesario consumo de energía y además posibles retrasos asociados. Por otro lado prevenir y evitar las colisiones puede producir una sobrecarga sustancial en la red, lo que deriva en un consumo mayor energía. La solución intermedia es uno de los factores clave.

2. A su vez, la transmisión de sobrecarga en el protocolo debe ser reducida tanto como sea posible, lo que incluye los paquetes dedicados al control de la red y los bits de cabecera de los paquetes de datos.

3. En los sistemas inalámbricos típicos, la radio del receptor ha de estar encendida siempre dando como resultado un aumento del consumo de energía. De nuevo, esto es más importante en una radio de corto alcance que en un sistema inalámbrico típico, como una red de computadores o telefonía móvil, ya que un gran porcentaje de la energía consumida ocurre cuando la radio está encendida. La situación ideal sería que la radio se encendiese sólo cuando el nodo necesite enviar o recibir paquetes sin que ello conlleve un aumento en los esfuerzos de monitorización.

4. Hay puntos fundamentales en el diseño de sistemas inalámbricos entre estos parámetros: eficiencia del uso de ancho de banda, retraso, calidad del canal, consumo de energía...

5. El último, pero no menos importante principio, es la capacidad de adaptación y movilidad. Para un sistema de gran escala con movilidad limitada, sería bueno un algoritmo adaptativo que funcionase con el mínimo sobrecoste posible.

3.4 Protocolo SMAC

3.4.1 Introducción

Sensor MAC es una capa que se encuentra entre la capa física y la capa de enrutamiento, además es el primer protocolo de control de acceso al medio (MAC) diseñado para Redes de Sensores Inalámbricos. Como hay que tener en cuenta que los nodos sensores están limitados por el consumo de energía, la idea básica, cuando fue creado este protocolo, es la necesidad de ahorrar energía. Por tanto, aparecen periodos en los que las radios de los nodos están encendidos y otros en los que están apagados.

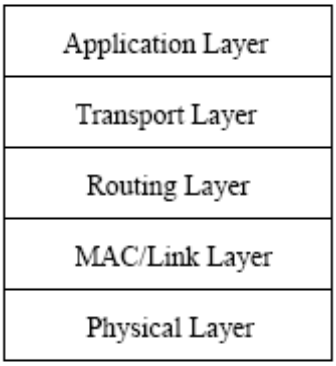

Figura 16: Modelo genérico por capas para WSN

Si hacemos un zoom sobre las capas física y MAC, podemos ver más detalles de las mismas, como ocurre en la siguiente figura.

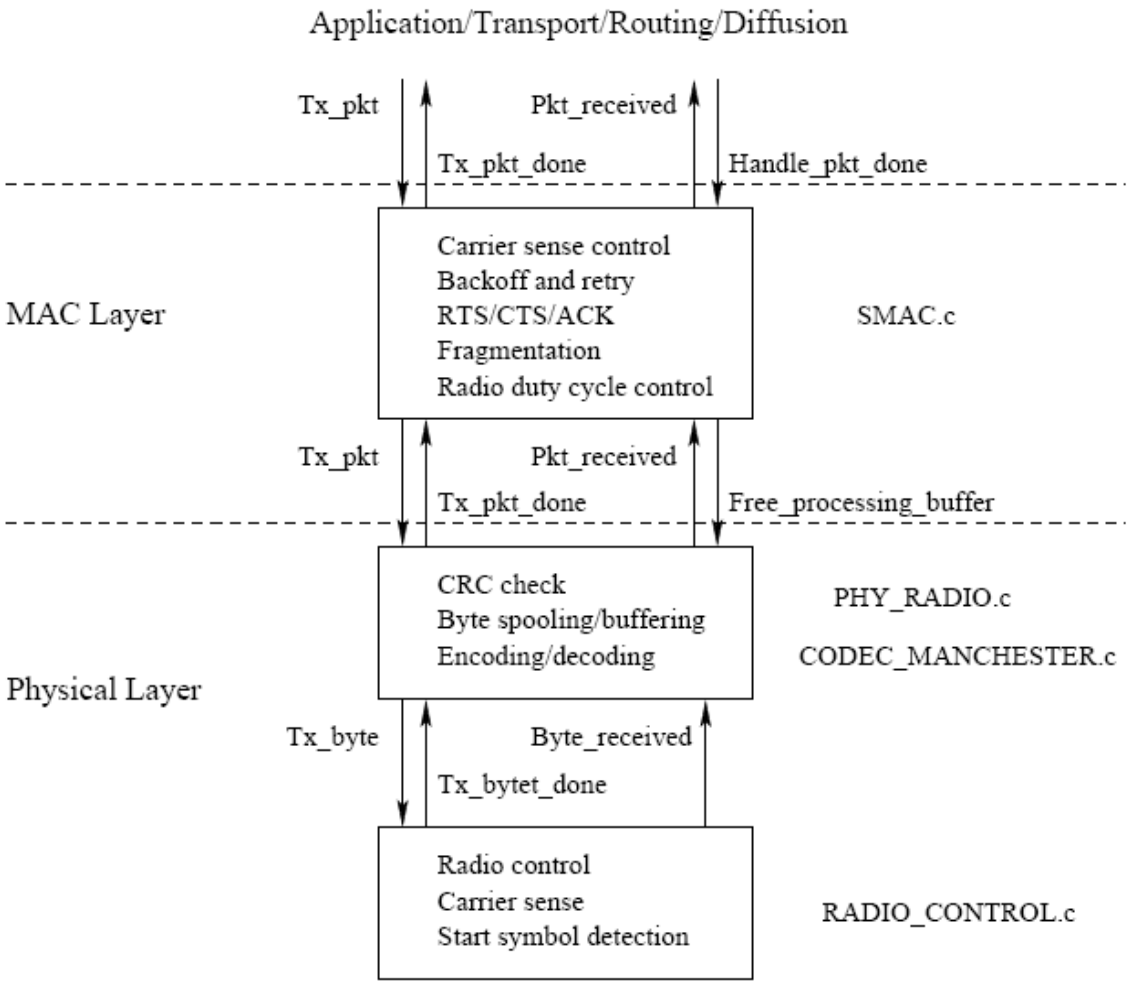

Figura 17: Capas física y MAC

Cuando existe una baja tasa de envío y una alta latencia, el protocolo es eficiente energéticamente, por estar los receptores durmiendo durante mucho tiempo.

La sincronización es necesaria en un protocolo donde los nodos sensores dependen unos de otros para establecer sus tiempos de radio encendida y radio apagada. Para conseguir dicha sincronización, existe un intercambio, entre los vecinos, de mensajes de sincronización SYNC.

Figura 18: Etapas del protocolo S-MAC

Otro problema a cubrir es el de la terminal escondida, por lo que se empleará la secuencia de paquetes RTS-CTS-ACK (Request to Send – Clear to Send - ACKnolegement). A través del paso de mensajes, un nodo podrá transmitir un conjunto de paquetes usando una única secuencia RTS/CTS para negociar y un paquete ACK para reconocer la llegada del paquete de datos. De esta manera, un paquete que reciba el mensaje RTS, sabrá la longitud del paquete que va a recibir y, por tanto, el momento en el que podrá dormir.

3.4.2 Mecanismo de control de potencia de transmisión

El mecanismo de control de potencia de transmisión (TPC) en la capa MAC de las Redes de Sensores Inalámbricos es un método para ahorrar energía. Los paquetes de datos serán transmitidos con la potencia mínima requerida y los demás paquetes, de control, serán transmitidos a la potencia nominal (máxima potencia).

Este mecanismo podrá ser implementado en el hardware de los actuales sensores y puede ser directamente aplicado a muchos protocolos MAC ya propuestos para las WSN.

El mayor gasto energético en los nodos se da en la comunicación vía radio. La consumición de la interfaz radio depende de su estado, el cual puede ser uno de los siguientes:

- **Transmission state**: transmisión del paquete donde la consumición de potencia es proporcional a la potencia de radio transmitida y puede ser seleccionada de entre un conjunto de valores discretos (en caso de las motas Mica2, de -20 a 5 dBm). Sin embargo, en la práctica la potencia de salida se mantiene en un alto valor fijo.
- **Reception -and listening- state**: recepción del paquete y escucha del canal (detecta portadora). Se consume una significante suma de potencia tanto al recibir los datos como escuchando (en las motas Mica2, son 35.4 mW).
- **Sleep state**: radio apagada. Existe una poca consumición de potencia en este estado (alrededor de 3 μW en los Mica2).

Desde la perspectiva de la capa MAC, la consumición puede ser minimizada si los nodos duermen durante la inactividad en vez de encontrarse en el estado de recepción. Por tanto, la consumición media será significativamente reducida. Dicha estrategia requiere la coordinación entre los nodos (todos ellos deben dormir y despertarse al mismo tiempo).

Pero el método que se usará en este proyecto es reduciendo la potencia sólo para

la transmisión de datos (manteniendo la misma potencia para los paquetes de control).

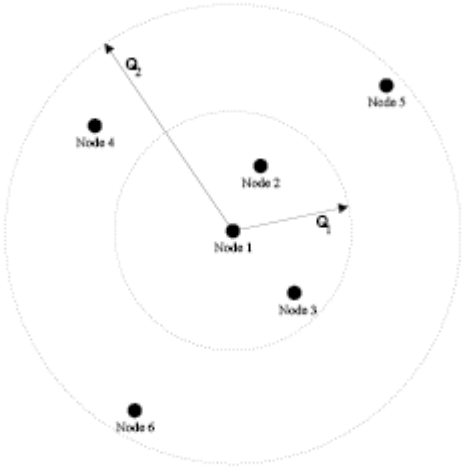

Figura 19: A diferentes distancias, distintas potencias de transmisión

En la anterior figura, el nodo 1 alcanza a los nodos 2 y 3 con la potencia Q_1, sin embargo los nodos 4, 5 y 6 sólo podrán ser alcanzados con la potencia Q_2. En este caso, Q_2 es la potencia nominal que garantiza la conectividad completa. Pero si la potencia es fija, existirá un gasto de potencia innecesario cuando el nodo 1 envíe datos a los nodos 2 y 3.

En este proyecto, se ha modificado la potencia de salida, por defecto, del protocolo Sensor MAC para que los paquetes de control (secuencia RTS-CTS-ACK) sean transmitidos a la potencia nominal (máxima). Y con la implementación del mecanismo de Control de Potencia Transmitida, los datos serán enviados a la mínima potencia necesaria para que lleguen a su destino.

La meta de este mecanismo es seleccionar el nivel de potencia de transmisión óptimo para ser empleado en el envío de cada paquete. Dicho nivel dependerá del alcance y del objetivo. Los protocolos TPC han sido comúnmente diseñados para las Redes Móviles Ad-hoc (MANET's) con el propósito de incrementar la capacidad en medios inalámbricos por medio de la reutilización del canal o para asegurar la conectividad de la red. Dos tipos de estrategias ya han sido consideradas:

(1): En la capa de red, TPC es usado para elegir el mejor subconjunto de los actuales vecinos para ser alcanzados.

(2): En la capa MAC, la potencia es seleccionada para mejorar la reutilización del canal para cada paquete o para reducir la probabilidad de colisión de los paquetes.

Las tres últimas propuestas están basadas en mecanismos muticanal, usando los canales adicionales para señalizar las transmisiones entrantes y calcular la mejor potencia

de salida para ser usada. Además, el protocolo TPC debe ser ejecutado en hardware que permita las variaciones muy rápidas de potencia de salida.

Estas soluciones consiguen reducir las colisiones y mejorar la capacidad en MANET's. Sin embargo, en WSN, la principal preocupación es la eficiencia de la consumición de energía. Además, los protocolos en WSN están limitados por los recursos disponibles de la memoria, la CPU y la radio. Por lo tanto, lo desarrollado para MANET's no es aplicable a las Redes de Sensores Inalámbricas.

3.5 Aplicación SMACTest

Esta aplicación testea la funcionalidad básica del protocolo S-MAC. La configuración, por defecto, es para ser usada con 3 nodos sensores. Cada nodo envía 10 mensajes broadcasts y 10 mensajes unicasts, cada uno con 5 fragmentos.

La configuración anterior se puede configurar en el archivo config.h. También se puede modificar S-MAC para ser ejecutado en diferentes modos: “low duty cycle with adaptative listen”, “low duty cycle without adaptive listen” y “fully active mode”.

También se puede modificar config.h para que sólo envíe mensajes broadcast o mensajes unicast. Al igual que puede cambiarse el tamaño de los mensajes mediante la variable “PHY_MAX_PKT_LEN”.

En el siguiente cuadro, pueden ver las configuraciones que, en el archivo config,h, pueden modificarse.

```
// Define la longitud máxima del paquete
// Los valores, por defecto, se encuentran en PhyMsg.h
// ------------------------------------------------------------
//#define PHY_MAX_PKT_LEN 50       // max: 250 (bytes), default: 100


// carrier sense threshold to determine a busy channel
// smaller value -> higher threshold -> more aggressive Tx
// this is only for mica2 and mica2dot
//#define RADIO_BUSY_THRESHOLD 0xb5  // 0x60 - 0xff, default: 0xb5


// Configure S-MAC into different operating modes
// ----------------------------------------------
//#define SMAC_DUTY_CYCLE 50       // 1 - 99 (%), default: 10
//#define SMAC_NO_ADAPTIVE_LISTEN  // default: adaptive listen is enabled
//#define SMAC_NO_SLEEP_CYCLE      // default: low duty cycle mode



// With the following macro defined, the node is configured as a slave on
// sleep schedules. It keeps listening to SYNC packets until it receives
// one and adopts it. With one master node and all other nodes as slaves,
// its easy to set up a network with only the master's schedule.
//#define SMAC_SLAVE_SCHED


// User adjustable S-MAC parameters
// --------------------------------
```

```
// Definitions here override default values in SMACConst.h

//#define SMAC_MAX_NUM_NEIGHB 20  // default value 20
//#define SMAC_MAX_NUM_SCHED 4    // default value 4
//#define SMAC_RTS_RETRY_LIMIT 7  // default value 7
//#define SMAC_DATA_RETX_LIMIT 3  // default value 3


// the following macro defines the maximum time that S-MAC can hold a message
// for transmission. If it cannot send out the message within the time, it
// will drop the message and signal upper layer tx failure.
// default value: 2min in low duty cycle mode and 10s in fully active mode
//#define SMAC_MAX_TX_MSG_TIME 120000


// The following two macros define the period to search for potential
// neighbors on different schedules. The period is expressed as the
// number of SYNC_PERIODs (10s). Therefore, 30 means after every 30
// SYNC_PERIODs, which is 300s, the node will keep listening for an entire
// SYNC_PERIOD. Maximum value is 255.
// The SHORT_PERIOD is used when I have no neighbor -- search more
aggressively
// The LONG_PERIOD is used when I have neighbors -- don't perform too often
//#define SMAC_SRCH_NBR_SHORT_PERIOD 3  // max: 255, default: 3  (30s)
//#define SMAC_SRCH_NBR_LONG_PERIOD 30  // max: 255, default: 30 (300s)


// Now by default, S-MAC uses timer/counter 0, which conflicts with Clock
// and Timer components. If you want to use Timer or Clock, uncomment the
// following line to let S-MAC use the 16-bit timer/counter 1
#define SMAC_USE_COUNTER_1


// define the following macro to put a time stamp on each outgoing packet
//#define SMAC_TX_TIME_STAMP
// TST_MSG_INTERVAL controls how fast a node generates a message. Setting
// it to 0 makes it generates second packet right after the first is sent.


// Configure the test application
// ------------------------------
#define TST_MIN_NODE_ID 1       // at least 2 nodes. node IDs must be
#define TST_MAX_NODE_ID 3       // consecutive from min to max
#define TST_MSG_INTERVAL 500    // in ms


// By default, each node keeps sending until it is powered off.
// To let a node automatically stop after sending sepecified number of
// messages, define the following macro
//#define TST_NUM_MSGS 20
```

```
// number of fragments in each unicast test message, max: 8
#define TST_NUM_FRAGS 1


// By default, each node alternate in sending broadcast and unicast
// for unicast, node i sends to (i+1), and node MaxId sends to MinId
//#define TST_BROADCAST_ONLY      // test broadcast only if defined
#define TST_UNICAST_ONLY        // test unicast only if defined
#define TST_UNICAST_ADDR 1      // specify unicast addr instead of default one
//#define TST_RECEIVE_ONLY        // set a node that only receives packets


// S-MAC debugging with a snooper
// ----------------------------
// Debug by adding bytes to data pkts, so that snooper can show them
//#define SMAC_SNOOPER_DEBUG


// The following macros are mutally exclusive. You can only define one
// Debugging with predefined S-MAC states and events
//#define SMAC_UART_DEBUG_STATE_EVENT
// Debugging by sending a byte to UART
//#define SMAC_UART_DEBUG_BYTE
```

Uno de los problemas que se tuvo al programar la aplicación “SMACTest” y que, posteriormente, se tuvo que tener en cuenta para el correcto funcionamiento de nuestra red fue definir las siguientes macros con los valores que, más abajo, aparecen:

```
#define TST_MIN_NODE_ID 1
#define TST_MAX_NODE_ID 2
```

lo cual hacía entender a la configuración de la aplicación que sólo habría un nodo en la red, creando un conflicto en la transmisión y recepción de paquetes. Por tanto, para utilizar 2 nodos, que han sido los necesarios para verificar la implementación del mecanismo TPC, se ha usado la siguiente configuración:

```
#define TST_MIN_NODE_ID 1
#define TST_MAX_NODE_ID 3
```

La última línea nos va a permitir la depuración mediante la aplicación “hyperterminal”, lo cual se explicará, de manera más minuciosa en el apartado 5.5 Depuración.

La aplicación SMACTest, durante su ejecución, activa o desactiva el LED amarillo para mostrar actividad (ya sea por envío de paquetes, recepción de paquetes, transmisión y recepción de paquetes SYN, ...). El LED rojo cambia de estado cuando un paquete o fragmento en enviado. El LED verde cambia de estado cuando un paquete o fragmento es recibido.

En este proyecto, las variables se han visto cambiadas dependiendo de la mota. Se

han usado 2 nodos sensores, uno que enviase mensajes unicasts solamente (con el máximo tamaño y sin dividirse en fragmentos) y otro que sólo recibiese dichos mensajes. De manera que, tras un largo periodo de tiempo, se pudiese comprobar el consumo energético generado por el nodo.

En el caso del nodo transmisor, las variables que se han cambiado en “config.h” han sido las siguientes:

```
#define TST_NUM_FRAGS 1
#define TST_UNICAST_ONLY
```

De manera que el paquete no se dividiese en fragmentos y que el envío de paquetes SÓLO fuese Unicast.

Para el caso del nodo receptor, las variables cambiadas han sido las siguientes:

```
#define TST_NUM_FRAGS 1
#define TST_UNICAST_ONLY
#define TST_RECEIVE_ONLY
```

Que se diferencia del nodo transmisor en que sólo puede recibir paquetes.

Por defecto, la aplicación está programada para que se envíen mensajes unicast y broadcast y éstos sigan el siguiente orden:

- nodo 1 envía mensajes al nodo 2
- nodo 2 envía mensajes al nodo 3
- nodo 3 envía mensajes al nodo 1

Pero con nuestra modificación, sólo el nodo 1 enviará mensajes al nodo 2.

Con respecto a la compilación de la aplicación, hay que tener en cuenta que se debe especificar la ID del nodo usando 'MIB510=/dev/ttyS0 make install.x mica2', donde x es la ID del nodo o especificándola en la variable TOS_LOCAL_ADDRESS en system/tos.h.

En el fichero config.h de la aplicación SMACTest, existe la variable siguiente variable:

```
//#define TST_UNICAST_ADDR 2      // specify unicast addr instead of default one
```

Un error que nos costó varias semanas de trabajo fue el siguiente:

Dicha variable se configuraba con la ID del nodo (para el transmisor, usábamos la ID 1 y para el receptor, la ID 2). Pero, con la configuración por defecto, el resultado de la aplicación no era el correcto.
Al darnos cuenta de dicho error, comprobamos el estado de los nodos sensores, los cuales funcionaban correctamente con otras aplicaciones. También nos pusimos en contacto con el autor del protocolo S-MAC (Wei Ye), quien nos advirtió de que se debería a alguna configuración o modificación que habíamos hecho en el código de la aplicación.

Finalmente, descubrimos que no es posible modificar la ID de los sensores cambiando dicha variable.

Una vez se han hecho los cambios comentados anteriormente, se puede calcular el gasto energético del nodo que transmite los paquetes unicast y sin emplear el mecanismo TPC. El siguiente paso es modificar el código, en la capa S-MAC, para emplear el mecanismo TPC.

Capítulo 4: Modificación del código

4.1 Introducción

La idea principal de este proyecto es la implementación del mecanismo de Control de Potencia Transmisión (TPC) en el protocolo Sensor MAC (S-MAC) y verificar que existe un ahorro de energía.

El mecanismo de Control de Potencia Transmitida envía los datos con la mínima potencia necesaria y los paquetes de control (secuencia RTS-CTS-ACK) a la potencia nominal.

Los nodos sensores que utilizamos son los Mica2. Éstos se ejecutan con el Sistema Operativo TinyOS y usan el protocolo S-MAC en la capa MAC. Además, se utilizará la aplicación “SMACTest” que acompaña al protocolo S-MAC y que puede encontrarse en el fichero \tinyos\cygwin\opt\tinyos-1.x\contrib\s-mac\apps. A continuación, se explicará la estructura desde la que se parte.

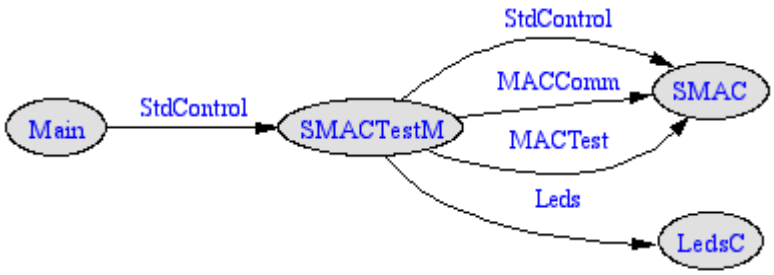

Figura 20: En la capa de aplicación, el componente SMACTest. En la capa MAC, el componente SMAC.

Cualquier componente que quiera ser una aplicación, deberá proveer la interfaz “StdControl”, como ocurre en la anterior figura. A la vez, la aplicación “SMACTest” está concatenada al componente “LedsC”, lo cual le servirá para encender y apagar los diodos leds de la mota.

El siguiente nivel, por debajo de la capa de aplicación, es el de la capa MAC (Sensor MAC). Para ello, el componente SMAC deberá proporcionar las interfaces “StdControl”, “MACControl” y “MACTest” y SMACTest usarlas.

Si analizamos las siguientes capas, nos encontraremos con el siguiente gráfico.

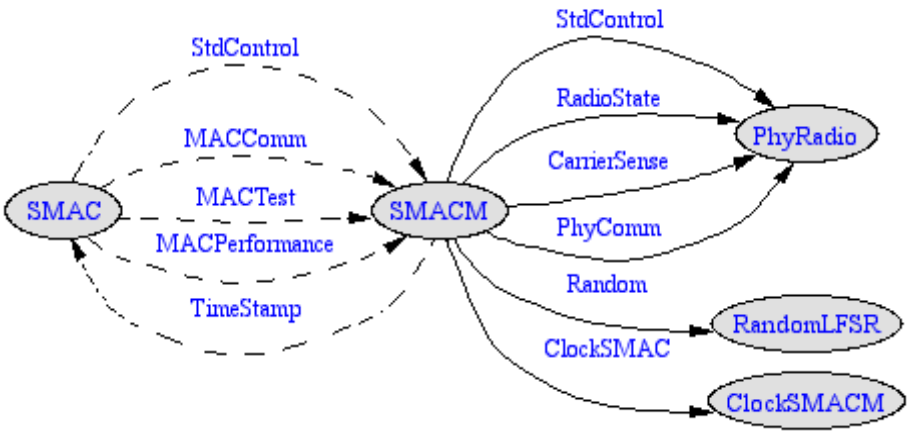

Figura 21: Capas que siguen a la MAC

SMAC contiene las secciones de 'Configuration' y de 'Implementation' y SMACM contiene la sección 'Module', que es en donde, realmente, se programa lo que hará el protocolo Sensor MAC. Las flechas discontinuas significan que tanto SMAC como SMACM proveen o requieren dichas interfaces. En el caso de la interfaz 'TimeStamp', ambos la requieren. En los demás casos, tanto SMAC como SMACM la proporcionan.

En la parte derecha del gráfico, puede observarse que la capa MAC está unida a la capa física de la radio, que sirve para enviar y recibir paquetes.

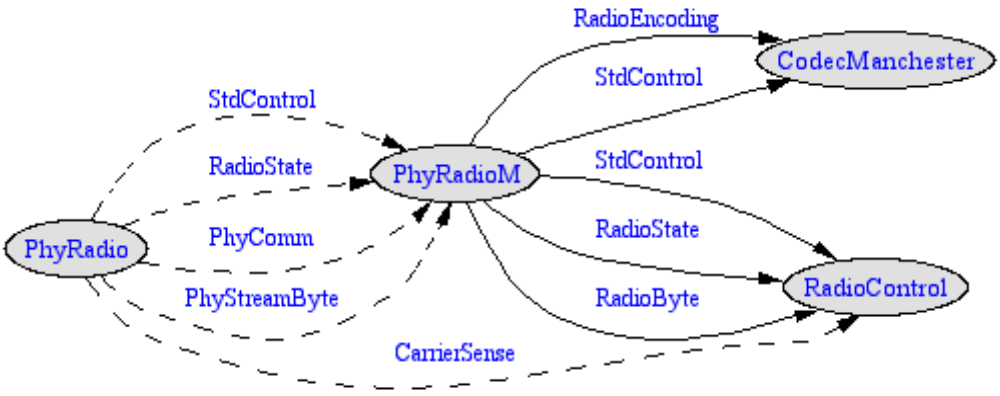

Figura 22: Capa radio

En el anterior gráfico, la capa física de la radio está unida a la capa de control de la radio. El módulo del componente PhyRadio requiere las interfaces “RadioEncoding” y “StdControl”, las cuales son proporcionadas por el componente “CodecManchester” para que la codificación sea Manchester.

La *codificación Manchester*, a la que también se le llama **codificación bifase-L**, es un método de codificación eléctrica de una señal binaria en el que en cada tiempo de bit hay una transición entre dos niveles de señal. Es una codificación autosincronizada, porque en cada bit se puede obtener la señal de reloj, lo que hace posible una sincronización precisa del flujo de datos. Una desventaja es que consume el doble de ancho de banda que una transmisión asíncrona.

Hoy en día hay numerosas codificaciones (8B/10B) que logran el mismo resultado pero consumiendo menor ancho de banda que la codificación Manchester.

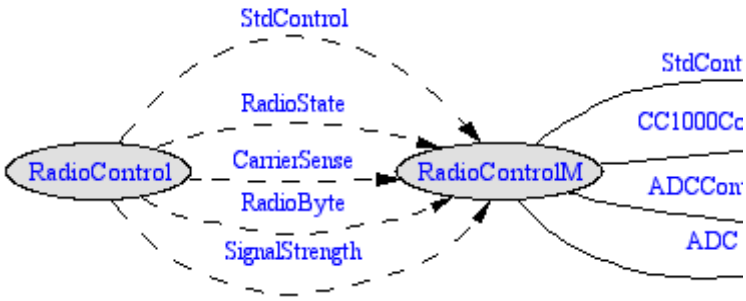

Figura 23: Componente RadioControl

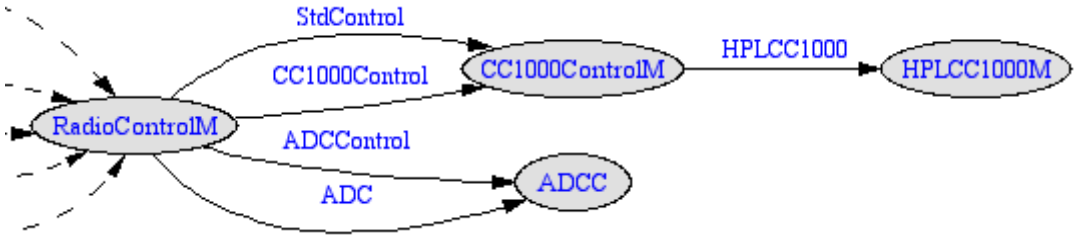

Figura 24: Nivel más bajo de la pila de protocolos

El componente RadioControl está unido a CC1000Control, que proporciona las funciones de CONTROL de las radios de la serie Chipcon1000. HPLCC1000 es el nivel más bajo de nuestra pila y es el que accede al chip CC1000.

4.1.1 Cygwin

Para la programación de los nodos sensores, se ha empleado un software llamado Cygwin y que se trata de una colección de herramientas desarrollada por la empresa Cygnus Solutions. Cygwin proporciona un comportamiento similar a los sistemas Unix en Windows. Su objetivo es portar software que ejecuta en sistemas POSIX a Windows con una recompilación a partir de sus fuentes.

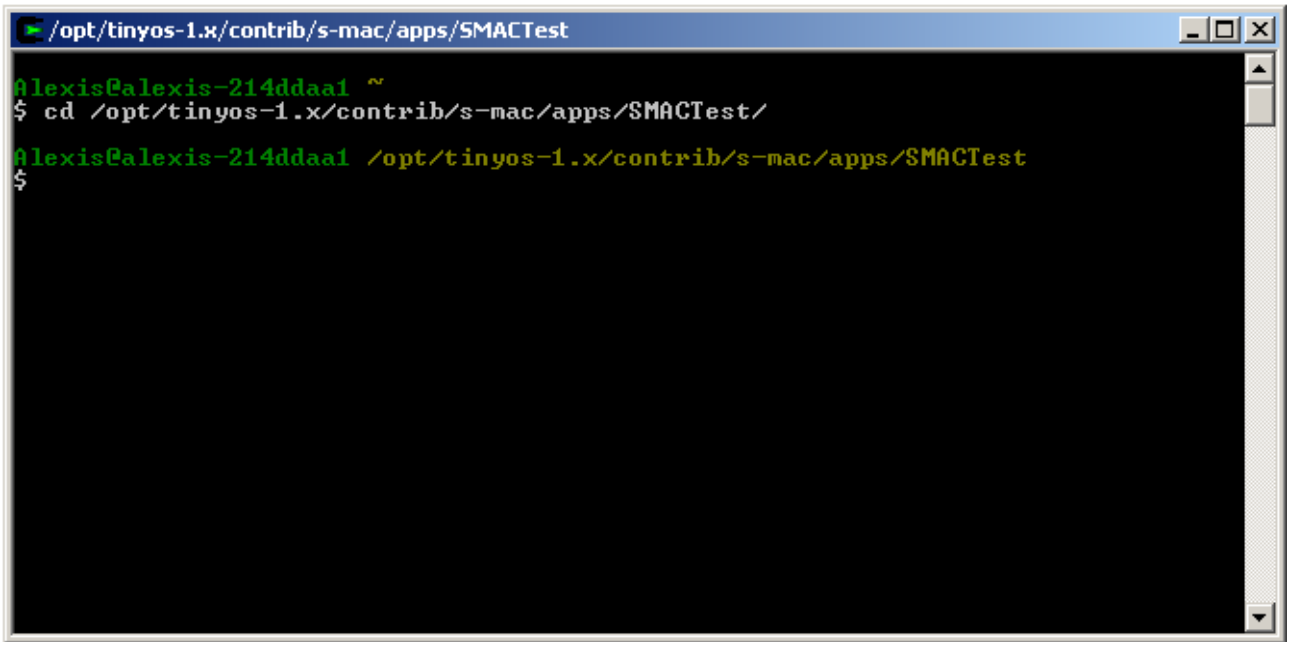

Figura 25: Ventana de Cygwin

En la actualidad, el paquete está mantenido principalmente por trabajadores de Red Hat. Se distribuye habitualmente bajo los términos de la GPL con la excepción de que permite ser enlazada con cualquier tipo de software libre cuya licencia esté de acuerdo con la definición de software libre. También es posible adquirir una cara licencia para distribuirla bajo otros tipos de licencia.

4.2 Modificación de la capa MAC para implementar el mecanismo TPC

En el envío de paquetes broadcast, nuestras motas se comunican mediante paquetes de control para asegurar que el canal está libre y que el paquete ha llegado correctamente. También existen paquetes de sincronización para ajustar las motas que se encuentran en la misma red. Los paquetes son los siguientes:

SYN	sincronización
RTS (Request To Send)	nodo 1 envía a nodo 2
CTS (Clear To Send)	nodo 2 responde a nodo 1
Datos	nodo 1 envía a nodo 2
ACK (ACKnolegement)	nodo 2 confirma la llegada de los datos

La capa MAC tiene varias funciones para enviar y recibir los paquetes anteriormente citados y son las presentadas a continuación:

- void **handleRTS** (void *pkt): identidfica un paquete RTS
- void **handleCTS** (void *pkt): identifica un paquete CTS
- void **handleDATA** (void *pkt): identifica un paquete de Datos
- void **handleACK** (void *pkt): identifica un paquete ACK
- void **handleSYNC** (void *pkt): identifica un paquete SYNC

- void **sendRTS** (void): envía un paquete RTS
- void **sendCTS** (void): envía un paquete CTS
- void **sendDATA** (void): envía un paquete de datos
- void **sendACK** (void): envía un paquete ACK
- void **sendSYNC** (void): envía un paquete SYNC

Además de las anteriores, existen otras funciones necesarias para la comunicación pero que no serán necesarias modificarlos para conseguir el ahorro energético.

Algunas de esas funciones son:

- command result_t **MACComm.broadcastMsg** (void *data, uint8_t length)
- void **startBcast** (void)

La idea clave de la modificación es leer el valor de la fuerza de la señal que le llega al transmisor (por tanto, habrá que leer la fuerza de la señal del paquete CTS recibido) y dependiendo de su valor, transmitir los datos a una potencia específica. Una vez se haya enviado el paquete, el valor de la potencia con que se transmitan los siguientes paquetes será la nominal (máxima).

4.2.1 Lectura de la relación Señal a Ruido del paquete CTS recibido

El chip CC1000, que poseen los mica2, tiene un pin que nos indica el RSSI (Received Signal Strength Indicator). Los bits IF_RSSI en el registro FRONT_END habilitan el RSSI. Cuando la función RSSI está habilitada, la corriente de salida del pin es inversamente proporcional al nivel de señal de entrada. La salida debería tener una resistencia, al final, para convertir la corriente de salida e un voltaje. Un condensador es usado para filtrar paso-bajo la señal.

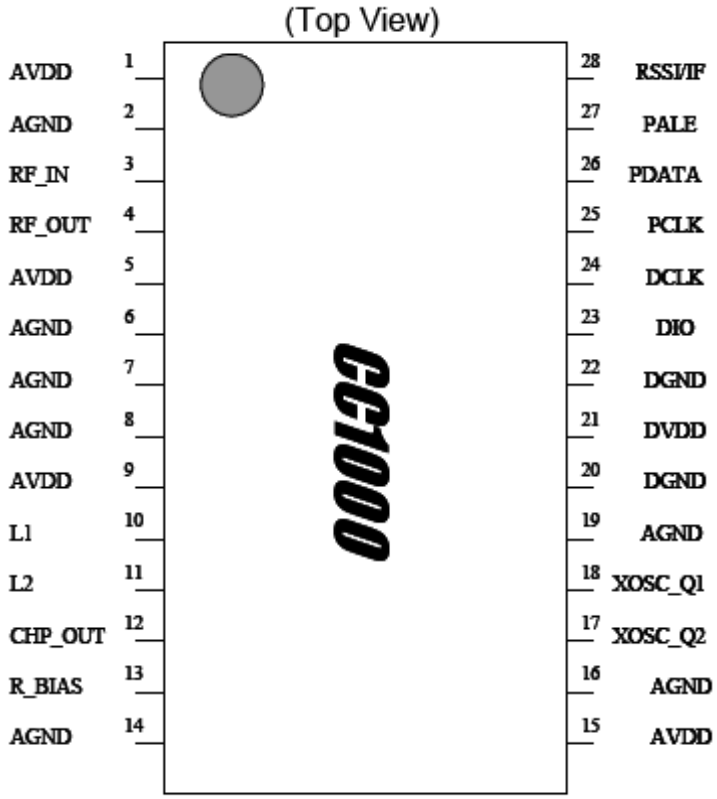

Figura 26: Chip CC1000

El rango de voltaje del RSSI es 0 – 1.2 V, usando una resistencia de 27 kΩ y dando, aproximadamente, 50 dB/V. El voltaje de RSSI puede ser medido por un conversor Analógico / Digital. Hay que tener presente que un valor muy alto de voltaje es una señal de entrada más baja.

El RSSI mide la potencia referida al pin RF_IN. La potencia de entrada puede ser calculada usando las siguientes ecuaciones:

P = -51.3 VRSSI– 49.2 [dBm] at 433 MHz
P = -50.0 VRSSI– 45.5 [dBm] at 868 Mhz

y como nuestras motas funcionan a 433 Mhz, tendremos que emplear la primera ecuación.

A continuación, se muestra la red externa para la operación del RSSI, siendo R281=27 kΩ y C281=1 nF.

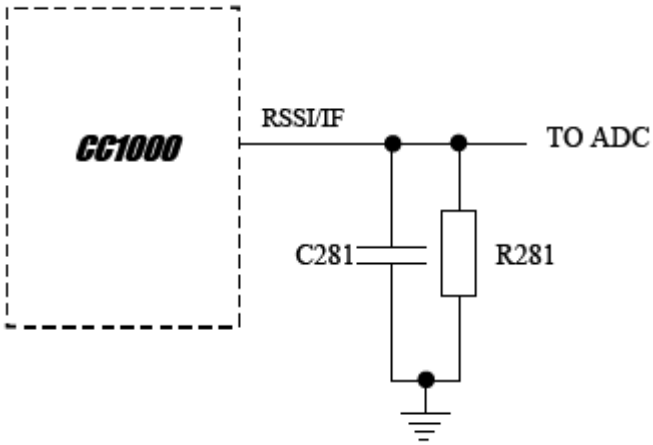

Figura 27: Circuito RSSI

La función encargada de recibir el paquete CTS del receptor en el nodo 1 es

void **handleCTS** (void *pkt)

Ésta se encuentra en el archivo “SMACM.nc” y en ella se tomará la relación Señal a Ruido, que es la diferencia entre el RSSI (Receive Signal Strength Indication) medido a través del canal 0 del conversor Analógico-Digital y el valor del ruido que se mide en la capa física de nuestra pila. Esto se puede traducirse en el código de la siguiente manera:

```
MACCtrlPkt* packet;                              (1)
packet = (MACCtrlPkt*)pkt;                       (2)

rssi = ((PhyPktBuf*)packet)->info.strength;      (3)
noiseLevel = call PhyComm.getNoiseLevel();       (4)
```

Tanto “rssi” como “noiseLevel” son variables globales que se han declarado al principio de SMACM.nc. La variable “pkt” es el paquete CTS recibido del nodo 2.

La fuerza de la señal es cargada en la estructura PhyPktInfo, la cual está incluida en PhyPktBuf, donde el paquete recibido es cargado. Al igual que lo hace el comando **handleCTS** (void *pkt), otras funciones que reciben paquetes, podrían conseguir una referencia a PhyPktBuf-Struct.

Pero los valores obtenidos tienen que ser convertidos en dBm, por lo que se aplicarán las siguientes ecuaciones para el caso de un Mica2:

$$Vrssi = Vbatt*ADCCounts/1024$$

$$RSSIdbm = -51.3*Vrssi – 49.2\ [dBm]$$

siendo Vbatt = 3 V. (la batería que lleva incorporada la mota).

Traducido a código, sería lo siguiente:

```
senal = -51.3*3*((float)rssi/1024)-49.2;
ruido = -51.3*3*((float)noiseLevel/1024)-49.2;
snr = senal-ruido;
```

donde las variables “senal”, “ruido” y “snr” son números flotantes. Como necesitamos que dichos valores sean números enteros, haremos la siguiente conversión:

```
senalInt = senal;
ruidoInt = ruido;
snrint = snr;
```

donde las variables “senalInt”, “ruidoInt” y “snrInt” son ahora números enteros.

4.2.2 Modificación de la potencia de salida

En la función sendData(), hemos de cambiar la potencia de salida a la que nos interese. Para ello, tenemos que unir la capa MAC con la capa de control CC1000Control. En SMACM.nc tendremos que incluir una línea:

```
  uses {
interface StdControl as PhyControl;
interface RadioState;
interface CarrierSense;
interface PhyComm;
interface Random;
interface ClockSMAC as Clock;
interface TimeStamp;
```

```
interface PowerManagement;
interface Leds;
interface CC1000Control;
}
```

Lo cual indica que se va a usar la interfaz

\tinyos\cygwin\opt\tinyos-1.x\tos\platform\mica2\CC1000Control.nc

En el archivo SMAC.nc se producen los siguientes cambios:

```
implementation
{
 components SMACM, PhyRadio, RandomLFSR, ClockSMACM,
HPLPowerManagementM, LedsC, CC1000ControlM;

 StdControl = SMACM;
 MACComm = SMACM;
 LinkState = SMACM;
 MACTest = SMACM;
 MACPerformance = SMACM;
 MACReport = SMACM;
 TimeStamp = SMACM;

 // wiring to lower layers

 SMACM.PhyControl -> PhyRadio;
 SMACM.RadioState -> PhyRadio;
 SMACM.CarrierSense -> PhyRadio;
 SMACM.PhyComm -> PhyRadio;
 SMACM.Random -> RandomLFSR;
 SMACM.Clock -> ClockSMACM;
 SMACM.PowerManagement -> HPLPowerManagementM;
 SMACM.Leds -> LedsC;
 SMACM.CC1000Control -> CC1000ControlM;
}
```

En este caso, se especifica que se va a emplear el componente CC1000Control (el módulo que proporciona control sobre todo tipo de chips de la serie CC1000). Además, se especifica que el componente SMACM usará la interfaz CC1000Control del componente CC1000ControlM.

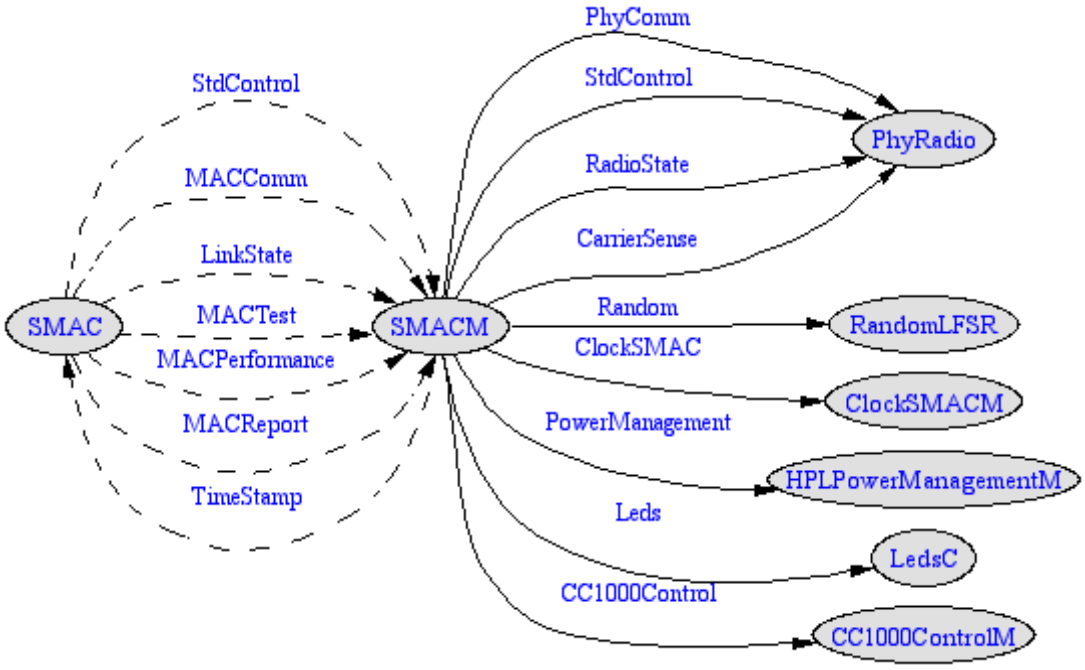

Figura 28: Nueva configuración de la pila de protocolos

Deben diferenciarse dos tipos de potencia: una será la nominal o máxima, a la que se envían los paquetes de control, y otra la calculada por nosotros como la mínima requerida para que al otro nodo le lleguen los datos. A continuación, se traduce a código las variables necesarias.

```
uint8_t newPower = 0xFF;
uint8_t defaultPower = 0xFF;
snDif = 0;                          // se explicara a continuación
```

La variable “newPower” es la potencia nueva a la que transmitir y que, por defecto, es la máxima (0xFF). La variable “defaultPower” es la potencia nominal, que siempre será máxima. La variable “snDif” indica una posición del array poTx[], que contiene la potencia a transmitir.

En primer lugar, habría que definir una relación Señal a Ruido umbral (que en nuestro código es llamada SMAC_SNR_UMBRAL) para que, a partir de la misma, se transmita cualquier paquete de datos (será la potencia mínima a la que transmitan los datos). Por defecto, el valor que se le asigna en SMACM.nc es '16'.

```
#ifndef SMAC_SNR_UMBRAL
#define SMAC_SNR_UMBRAL 16
#endif
```

Con esta S/N umbral, podemos obtener la siguiente diferencia:

snDif = [(S/N)real] – [(S/N)umbral]

donde [(S/N)real] es señal medida.

En segundo lugar, calcularemos la diferencia entre la potencia máxima y snDif, obteniendo la potencia a la que debemos transmitir nuestro paquete de datos.

Para modificar la potencia de salida, se dispone de 31 valores posibles y que están almacenados en el array poTx[] en orden descendente, es decir, la potencia que ocupa la posición 0 es la máxima. Como en cada posición del array se decrementa la potencia 1 dB, sólo será necesario acceder al array en la posición que indique el valor de snDif.

```
uint8_t poTx[] = {0xff, 0xe0, 0xc0, 0x90, 0x80, 0x70, 0x60, 0x50, 0x50, 0x40,
0x0f, 0x0e, 0x0c, 0x0b, 0x0a, 0x09, 0x08, 0x07, 0x06, 0x05, 0x05, 0x04, 0x04,
0x03, 0x03, 0x03, 0x02, 0x02, 0x02, 0x01, 0x01};
```

A continuación, a 433 Mhz, se reflejan los valores que puede tomar la potencia de salida, el valor en hexadecimal y el consumo de corriente.

Output power [dBm]	RF frequency 433 MHz	
	PA_POW [hex]	Current consumption, typ. [mA]
-20	01	5.3
-19	01	6.9
-18	02	7.1
-17	02	7.1
-16	02	7.1
-15	03	7.4
-14	03	7.4
-13	03	7.4
-12	04	7.6
-11	04	7.6
-10	05	7.9
-9	05	7.9
-8	06	8.2
-7	07	8.4
-6	08	8.7
-5	09	8.9
-4	0A	9.6
-3	0B	9.4
-2	0C	9.7
-1	0E	10.2
0	0F	10.4
1	40	11.8
2	50	12.8
3	50	12.8
4	60	13.8
5	70	14.8
6	80	15.8
7	90	16.8
8	C0	20.0
9	E0	22.1
10	FF	26.7

Tabla 12: Valores de la potencia de salida

Traduciendo a código el cálculo de snDif y el establecimiento de la nueva potencia, se obtiene lo siguiente:

```
if(snrint>SMAC_SNR_UMBRAL)
 snDif = snrint - SMAC_SNR_UMBRAL;

if(snDif>30)            // 30 dB es lo mínimo que ofrece CC1000
 snDif = 30;

newPower = poTx[snDif];
```

Una vez calculada la potencia a la que debemos transmitir, podremos modificarla mediante la siguiente llamada:

call CC1000Control.SetRFPower(newPower);

Una vez transmitido el paquete, que se conseguiría con el siguiente código, podremos establecer la potencia a su valor por defecto.

```
#ifdef SMAC_TX_TIME_STAMP
  dataPkt->txTimeStamp = clockTime;
#endif

// neighbNav tracks the time I have left for tx
 dataPkt->duration = neighbNav - durDataPkt;
 call Leds.redToggle();
 call PhyComm.txPkt(dataPkt, txPktLen);

 call CC1000Control.SetRFPower(defaultPower); //potencia por defecto
```

4.3 Modificación de la potencia de salida de los paquetes de control

Por defecto, el chip CC1000 envía todos los paquetes a una potencia menor que la máxima. Por tanto, necesitamos cambiar dicha configuración para comprobar el efecto del mecanismo TPC desde distancias muy cortas a distancias mucho más largas. Así como cumplir con las especificaciones del mecanismo de Control de la Potencia de Transmisión, en el que es una condición indispensable el enviar los paquetes de control a la máxima potencia.

Para alterar dicho valor, es necesario acceder a:

\tinyos\cygwin\opt\tinyos-1.x\tos\platform\mica2\CC1000Const.h

y cambiar la línea:

```
#define CC1K_PA_POW  0x0B   //11
```

por la siguiente otra:

```
#define CC1K_PA_POW  0xFF   //potencia nominal
```

4.5 Depuración

Para facilitar la modificación del protocolo S-MAC, ha sido necesario un mecanismo de depuración. A lo largo de la realización de este proyecto, se han usado dos medidas para verificar que nuestra aplicación se ejecutaba en los distintos puntos del código.

Por un lado, se han utilizado los diodos leds, los cuales pueden ser controlados gracias al componente LedsC. Las funciones disponibles para controlar los leds son las siguientes:

```
# async command result_t Leds.redOn (void)          //enciende el led rojo
# async command result_t Leds.redOff (void)         //apaga el led rojo
# async command result_t Leds.redToggle (void)      //cambia el estado del led rojo
# async command result_t Leds.greenOn (void)
# async command result_t Leds.greenOff (void)
# async command result_t Leds.greenToggle (void)
# async command result_t Leds.yellowOn (void)
# async command result_t Leds.yellowOff (void)
# async command result_t Leds.yellowToggle (void)
```

Posteriormente y debido a las limitaciones de este método de depuración, se empleó el mecanismo de depuración que incluye el protocolo S-MAC.

Para ello, hay que incluir la siguiente cabecera:

```
#include "smacUartDebug.h"
```

Para iniciar la depuración, hay que incluir en la iniciación de la capa MAC, la siguiente la línea:

```
smacUartDebug_init();
```

Para imprimir un byte en el hyperterminal, emplearemos la siguiente función:

```
static inline void smacUartDebug_byte(char byte);
```

Para ganar en claridad, se ha creado un macro-identificador llamado

SMAC_UART_DEBUG_TX

y que puede ser definido en el archivo config.h, de manera que sólo se depure cuando lo declaremos en dicho archivo. En SMACM.nc y mediante condiciones especializadas, se comprueba si la macro está definida. Si lo está, se procede a imprimir en hyperterminal los datos necesarios para la depuración.

Capítulo 5: Mediciones

5.1 Introducción

El objetivo de las mediciones era medir el consumo energético que los sensores tenían tras realizar la modificación del protocolo (adición del TPC a S-MAC). El objetivo era comparar estos resultados con los obtenidos por un sensor en las mismas condiciones pero con el protocolo S-MAC original. En principio esta medida se puede obtener de manera sencilla colocando un medidor de corriente (amperímetro) en serie con el aparato, tomando muestras durante un tiempo determinado y realizando luego los cálculos pertinentes:

$$I = \frac{V}{R};$$

$$P = V \cdot I;$$

$$E = P \cdot t;$$

Por lo que, una vez tomadas las muestras sólo queda multiplicarlas todas por la tensión de alimentación del aparato (como se indica en la segunda fórmula y por el tiempo que haya durado la prueba para obtener una medida de la energía consumida (ver la tercera fórmula). De este modo, aplicando esta prueba a dos sensores durante el mismo tiempo, uno con una aplicación de test que utilice S-MAC con TPC y otro sin TPC podemos obtener una relación que nos indique el ahorro energético producido. Para que la tensión suministrada fuese constante la mejor opción parecía alimentar el MICA2 con una fuente de alimentación para que, de este modo, la tensión fuese constante. Si hubiesemos dejado la alimentación original mediante baterías AA hubiera surgido un problema ya que a lo largo del tiempo, con el desgaste de la pila, la tensión suministrada decrece y, al no ser constante, el cálculo de la energía consumida se hace más complejo. Por este motivo decidimos utilizar una fuente de alimentación para proporcionar energía a los sensores y que éstos estuviesen alimentados de forma constante.

5.2 Problemas en las mediciones

Al principio, la idea para realizar las medidas era utilizar el osciloscopio disponible en el laboratorio. Dicho modelo dispone de una interfaz RS-232 que, conectada al ordenador y utilizando el software proporcionado por el fabricante, permite almacenar las muestras tomadas por el osciloscopio en un PC. Al probar el software y comenzar a utilizarlo descubrimos que el osciloscopio tenía un grave problema; a pesar de que toma muestras por segundo, a través de la interfaz serie sólo puede enviar, como máximo una cada 250 ms (4 muestras por segundo). Si tenemos en cuenta que los paquetes de datos de la aplicación de test ocupan 100 bytes (800 bits) y que el sensor transmite a 19,2 kbps:

$$t = \frac{800}{19,2 \cdot 10^3} = 0,041\overline{6} \cong 42ms$$

Si como se ha dicho, el osciloscopio envia muestras al PC, en el mejor de los casos cada 250 ms, las medidas tomadas mediante este método no reflejarán en absoluto el consumo real de los sensores pues, como se puede observar es posible que las medidas coincidan con el momento de una transmisión pero también es posible que no; eso sin tener en cuenta que los paquetes de control son mucho más pequeños (decenas de bits) y por supuesto, tampoco apareceran en las medidas.

5.2.1 Primera solución

Ante el problema surgido con el almacenamiento de muestras del osciloscopio, decidimos recurrir al uso de otro osciloscopio disponible en el laboratorio. En este caso, se trataba de un Yokogawa DL-1540, que es capaz de tomar hasta 20 millones de muestras por segundo (20 MS/s). Este osciloscopio dispone de una disquetera integrada que permite almacenar tanto muestras como capturas de cualquier medida realizada. Tras una pequeña investigación en el manual descubrimos que este osciloscopio cumple sobradamente la frecuencia de muestreo pero el problema surgía (como con el otro) a la hora de almacenar las muestras.

Este modelo concreto solo soporta el almacenamiento de 10.000 muestras consecutivas por lo que, a la hora de realizar las pruebas, utilizando una frecuencia de muestreo de 100.000 muestras por segundo (100 kS/s), la mayor captura posible es de 100 ms. Lógicamente, una prueba de consumo de 100 ms no es válida para determinar el ahorro energético. Por lo cual esta solución se declaró como no válida y se optó por buscar otra.

Figura 29: *Imagen de un osciloscopio Yokogawa DL-1540*

5.2.2. Segunda solución

Teniendo en cuenta que el principal problema era el almacenamiento de muestras para el posterior cálculo del consumo, se optó por usar la tarjeta de sonido de un ordenador como osciloscopio para la adquisición de datos. Al fin y al cabo, se trata de un conversor A/D que toma muestras a través de la entrada de línea y de la del micrófono. En principio pareció ser la solución perfecta ya que la frecuencia de muestreo (44,1 kHz) era suficiente y el problema del almacenamiento dejaba de existir, puesto que es posible importar la señal con cualquier programa de edición de sonido.

Aunque estos programas no indiquen el nivel de voltaje, la amplitud que marquen puede ser fácilmente ponderada realizando previamente algunas pruebas a tensiones conocidas (si introducimos una tensión de un voltio podemos tomar una referencia de la amplitud que indica el programa y ponderar los resultados obtenidos). Nos pusimos manos a la obra desmontando un conector Jack de 3,5 mm estéreo como el de la siguiente figura.

Figura 30: Conector Jack de 3.5 mm estéreo

Una vez desmontado, procedimos a conectar mediante pinzas el Jack al circuito, como se puede observar en la figura 31 se dispone de los dos canales y un tercer terminal para conectar a la masa.

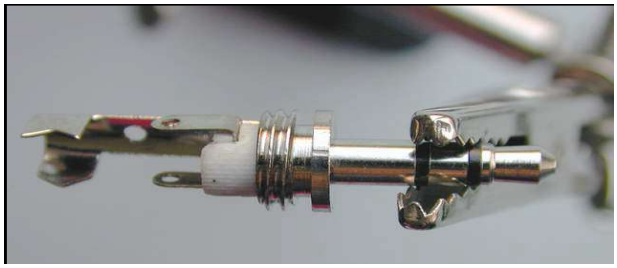

Figura 31: Conector Jack 3.5 mm desmontado

Decidimos realizar un sencillo circuito de test para probar el nuevo “osciloscopio”. Conectamos una resistencia de 1 en serie con un MICA2, estando éste alimentado por una fuente de tensión de 3V. De este modo, midiendo en los bornes de la resistencia obtendremos la caída de tensión que nos indicará la corriente que circula por el circuito (y utilizando las fórmulas del epígrafe 5.1 Introducción, podremos obtener la energía consumida). Procedimos a enchufar el conector Jack a la entrada de línea de la tarjeta de sonido y utilizamos un programa editor de audio de libre distribución llamado Audacity. Tras habilitar el micrófono desde el sistema operativo, desactivar el amplificador de 20dB que incorpora por defecto y comenzar la captura, encendimos la fuente de alimentación. El programa comenzó a capturar una señal que, efectivamente variaba con los períodos de transmisión de paquetes de la aplicación de test (dicha aplicación utiliza los LEDs para indicar los periodos de actividad y la transmisión/recepción de paqutes). El problema venía en la forma que el programa tiene de representar la señal; ya que los resultados habían sido presumiblemente satisfactorios decidimos investigar un poco más por internet. Encontramos bastantes programas distintos que sirven para simular un osciloscopio mediante la tarjeta de sonido. Después de probar un par de ellos, nos decidimos por Soundcard Oscilloscope, un software gratuito que permite su uso para fines educativos y no comerciales; su autor es el alemán Christian Zeitnitz y funciona perfectamente sobre la plataforma Windows, funcionando además como generador de señales (aunque esta función resulta irrelevante en nuestro caso).

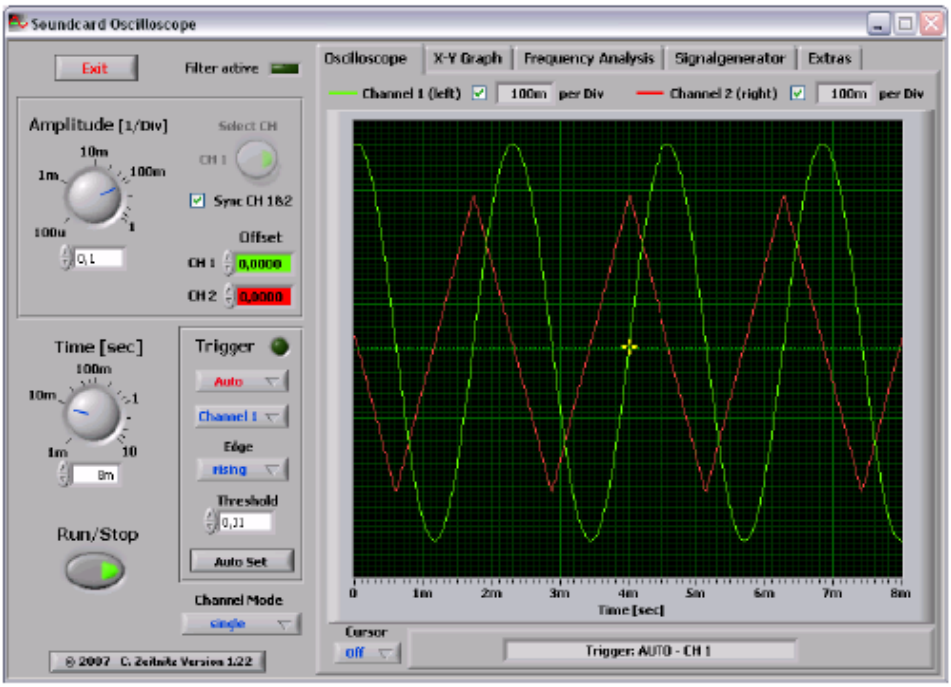

Figura 32: Captura del programa Souncard Oscilloscope funcionando sobreWindows XP

Repetimos la anterior prueba y esta vez la representación si era correcta, de hecho era aparentemente igual que la que habíamos obtenido con el anterior osciloscopio (ver figura 29). Tras observar durante un tiempo la representación nos dimos cuenta de que la señal, al estabilizarse se centraba en cero, es decir, los períodos en los que la tensión era constante, la caída de tensión que representaba el programa era 0V. Tras consultar diversas fuentes, descubrimos que las tarjetas de sonido incluyen un condensador a su entrada que se utiliza para filtrar la componente continua de las señales de entrada por lo que, una vez más, no pudimos utilizar esta solución.

5.2.3. Tercera y última solución

Tras el último fracaso decidimos probar una nueva solución; desempolvando los apuntes de Electrónica Analógica, recordamos un circuito que podía realizarnos, de manera autónoma, el proceso de almacenamiento de las distintas muestras, además, con mucha más precisión; se trata del circuito integrador.

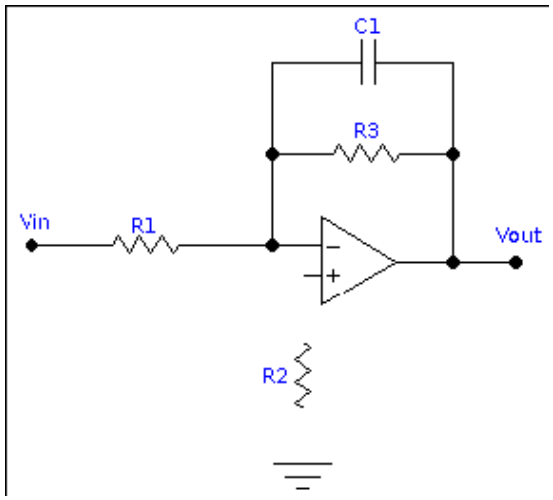

Figura 33: Circuito integrador realizado con un amplificador operacional

Dicho circuito, saca a su salida la integral de la señal que recibe a su entrada multiplicada por un factor dependiente de los valores de R1 y C1; la tensión de salida viene dada por la siguiente expresión:

$$V_{OUT} = -\frac{1}{R1 \cdot C1} \int_{t1}^{t2} V_{IN} dt$$

De manera que si conectamos a la entrada de dicho circuito la caída de tensión en una resistencia en serie con el sensor, y a la salida un osciloscopio que mida la tensión de salida (Vout) tendremos, en dicho osciloscopio la suma de todos los valores a la entrada del circuito. Si la resistencia es lo suficientemente baja (menor o igual que 10) no influirá en el consumo por lo que solo quedará multiplicar la tensión de salida (VOUT) por el factor mencionado (R1 • C1), dividirlo por la resistencia sobre la que se mide la caída y multiplicarlo por la tensión de alimentación:

$$V_{OUT} = -\frac{1}{R1 \cdot C1} \int_{t1}^{t2} V_{IN} dt$$

usando dos resistencias de 2,2M (R1 = R2 hace que disminuya el error de offset), un condensador de 10µF, una resistencia (R3) de 220, otra resistencia de 10 y tres fuentes de alimentación (±15V para alimentar el operacional y 3,3V para el MICA2).

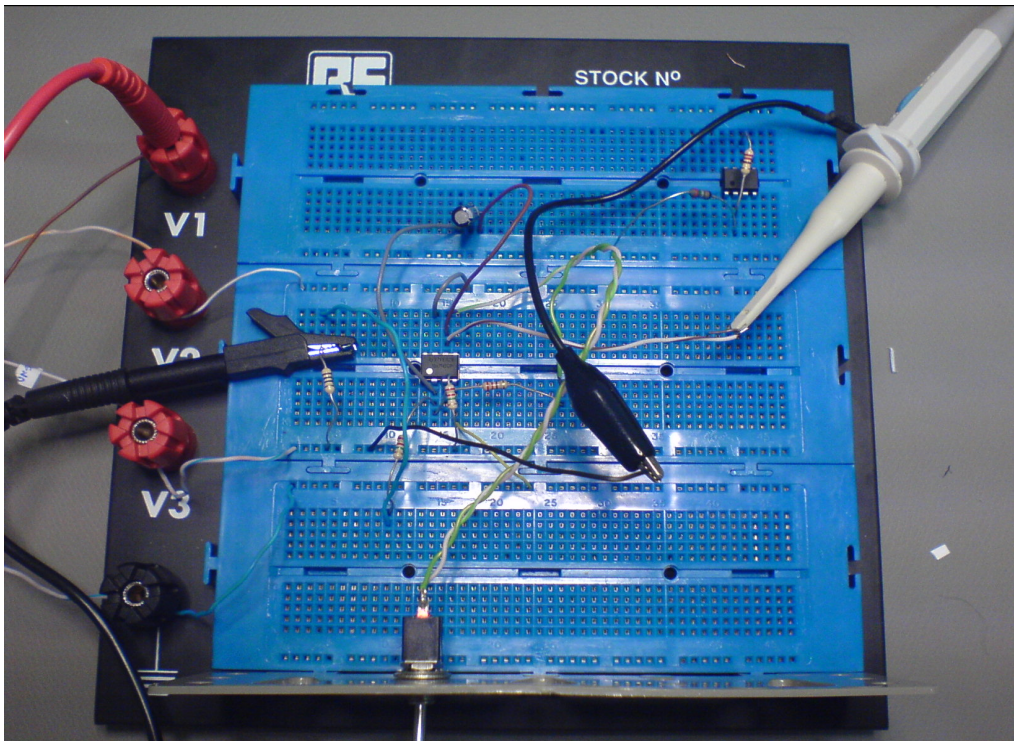

Figura 34: Circuito integrador montado en el laboratorio

Tras un buen rato montando y probando el circuito sobre la placa board, terminamos el diseño y quedó como muestra la figura anterior. Realizamos algunas pruebas con resistencias constantes para ver como se comportaba; cuanto tiempo tardaba en entrar en saturación, que tensión alcanzaba en dicha zona... Tras unas cuantas pruebas más, el circuito estaba listo para realizar su tarea, solo había un pequeño inconveniente, estábamos utilizando tres fuentes de alimentación distintas, lo cual era muy aparatoso y además, una de ellas era necesaria en el laboratorio ya que otro grupo debía usarla.

5.2.4. Fuentes de alimentación

Como ya se ha comentado, disponer de tres fuentes de alimentación era demasiado aparatoso, eso sin contar con el hecho de que, realmente, solo podíamos utilizar dos. Pensando las diversas opciones que teníamos, decidimos utilizar una fuente de alimentación de un PC. Tras conseguir el esquema de voltaje del conector de una fuente ATX descubrimos que, si conseguiamos utilizarla era perfecta ya que proporcionaba todos los niveles de tensión que nuestro circuito requería (±12V y 3,3V). Las fuentes de alimentación ATX en principio, no pueden conectarse solas, debe ser la placa base la que las 'encienda'. La placa base, cuando se presiona el pulsador de encendido, dispara un tiristor que queda conduciendo y "puentea" dos terminales de la fuente produciendo el encendido de la misma. Sabiendo esto, para poder encender la fuente, soldamos un interruptor a los dos pines de la fuente que había que unir para poder apagar y encender dicha fuente cuando fuese necesario.

Tras asegurar un poco las conexiones, etiquetar adecuadamente todos los cables y realizar algunas medidas de prueba, la fuente estaba totalmente lista para su uso con los MICA2 y el circuito de test.

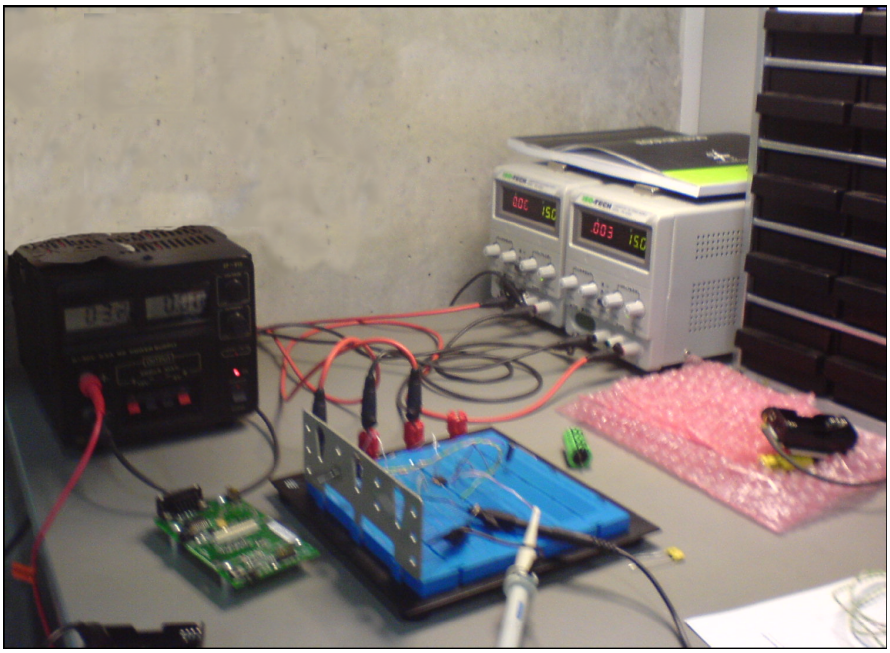

Figura 35: Montaje inicial realizado en el laboratorio con tres fuentes de alimentación

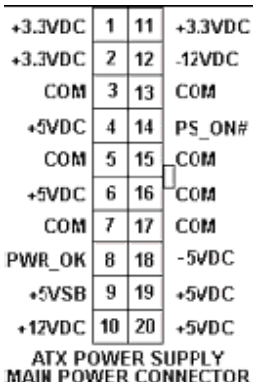

Figura 36: Esquema de voltajes del conector de una fuente ATX

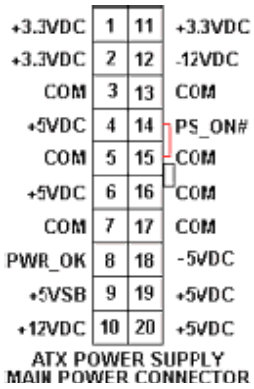

Figura 37: En rojo se pueden ver los dos pines que es necesario conectar para encender la fuente.

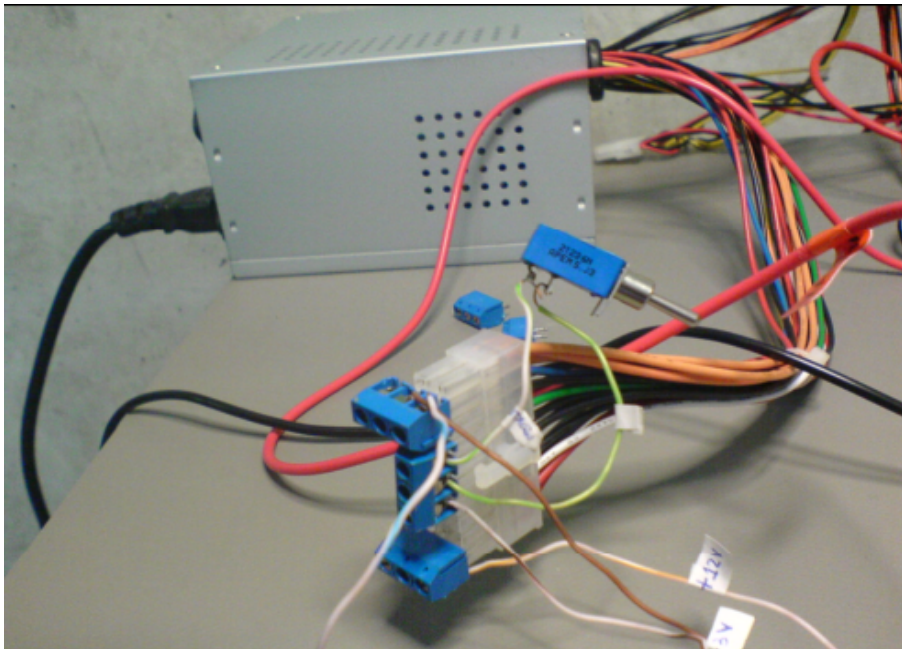

Figura 38: Conector ATX con las modificaciones realizadas para usarlo en el circuito de medidas

5.3. Mediciones

Las primeras medidas de consumo de los MICA2 se realizaron en el interior del laboratorio I+D IT2. Para empezar a probar, decidimos hacer unas pruebas iniciales que nos permitieran comprobar la aplicación de test y comenzar a comparar consumos respecto a la misma aplicación haciendo uso de S-MAC sin TPC. Para ello montamos una prueba en el laboratorio, separando los sensores 50 cm para dejarlos transmitiendo. De las primeras pruebas con el circuito integrador comprobamos que, con la tensión de alimentación que tenía el operacional (±12V) entraba en saturación, aproximadamente al

llegar la tensión de salida a los 9V. Sabiendo esto conectamos el sensor al circuito de medida y lo dejamos funcionar durante 5 min para ver, aproximadamente, qué pendiente tenía la función de salida del integrador, y con ello, comprobar cuán larga podíamos hacer la prueba sin que el amplificador operacional entrara en saturación.

Tras una primera estimación de la pendiente, para no arriesgar ante posibles cambios en un período más largo, decidimos hacer las pruebas de 25 min. de duración. Con esta duración realizamos, con la misma distancia, la medida de consumo para un sensor con TPC primero y después para otro sin él.

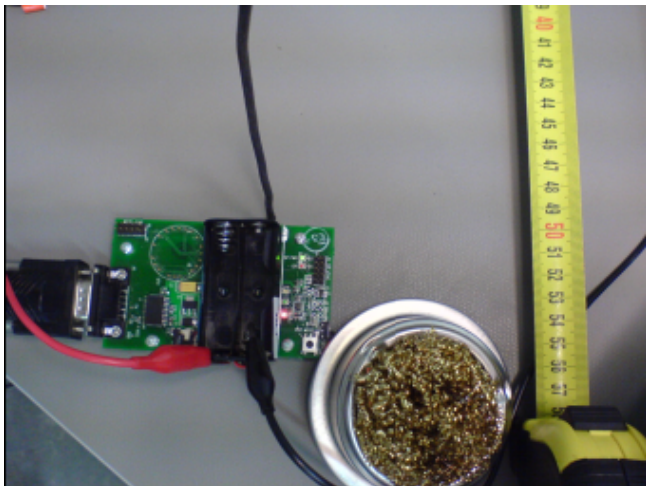

Figura 39: La primera prueba fue realizada a 50 cm de distancia entre nodos

Tras estas primeras medidas en el laboratorio, se hicieron otras en un espacio abierto (en el patio del Cuartel de Antigones) y se obtuvieron ahorros cercanos al 12%, lo cual era un buen ahorro. Tras hacer las pruebas a distancias mayores de 16 metros, descubrimos que los nodos se quedaban sin cobertura por estar al ras del suelo. Por tanto, se dieron las medidas por no válidas.

Las siguientes medidas, tras el fracaso de las anteriores, consistieron en colocar los nodos sensores encima de sillas y poder evitar el contacto directo con el suelo. Con estas mediciones, se concluyó que existía bastante ruido a la salida del integrador (alrededor de +/- 100 mV en media). Entonces, decidimos tomar como muestra el último mS y, empleando las funciones del osciloscopio, tomar la medida de ese último mS y los valores máximos y mínimos (para tener una idea de cómo el ruido hacía oscilar la señal). Incluso haciendo estas nuevas mediciones, descubrimos que la potencia consumida varía mucho de un día a otro (por posibles interferencias, el factor de corrección de potencia de la fuente de alimentación, ...).

Las últimas medidas tomadas, se presentan a continuación.

Empleando el mecanismo de TPC:

Tiempo (min)	Distancia (m)	Vout {Avg} (V)	Energía consumida (J)	Ahorro (%)
25,01000000	0,50000000	-4,92700000	40,16248675	3,15
25,01000000	0,50000000	-5,08600000	41,45857675	0,02
25,01000000	0,50000000	-4,73400000	38,58924544	6,94
25,01000000	0,50000000	-4,44800000	36,25791376	12,56
25,01000000	1,00000000	-4,16600000	33,95918811	18,49
25,01000000	1,00000000	-5,08800000	41,47487976	0,45
25,01000000	2,00000000	-5,01700000	40,89612260	2,9
25,01000000	4,00000000	-5,11300000	41,67866750	0,17
25,01000000	8,00000000	-5,30500000	43,24375730	-3,58
25,01000000	16,00000000	-5,20700000	42,44490938	-1,67
25,01000000	32,00000000	-5,19600000	42,35524278	-1,45
25,01000000	40,00000000	-4,96700000	40,48854713	3,02

Tabla 13: Valores con TPC

Distancia (m)	Vout {Avg} (V)	Maximo	Minimo	Energía consumida (J)	Error (V)	Error (J)
0,50000000	-4,92700000	-4,80000000	-5,04000000	40,16248675	0,24	1,95636225
0,50000000	-5,08600000	-5,04000000	-5,12000000	41,45857675	0,08	0,65212075
0,50000000	-4,73400000	-4,72000000	-4,80000000	38,58924544	0,08	0,65212075
0,50000000	-4,44800000	-4,40000000	-4,48000000	36,25791376	0,08	0,65212075
1,00000000	-4,16600000	-4,12000000	-4,24000000	33,95918811	0,12	0,97818113
1,00000000	-5,08800000	-5,08000000	-5,16000000	41,47487976	0,08	0,65212075
2,00000000	-5,01700000	-4,88000000	-5,12000000	40,89612260	0,24	1,95636225
4,00000000	-5,11300000	-5,00000000	-5,28000000	41,67866750	0,28	2,28242263
8,00000000	-5,30500000	-5,12000000	-5,40000000	43,24375730	0,28	2,28242263
16,00000000	-5,20700000	-5,08000000	-5,36000000	42,44490938	0,28	2,28242263
32,00000000	-5,19600000	-5,16000000	-5,24000000	42,35524278	0,08	0,65212075
40,00000000	-4,96700000	-4,92000000	-5,00000000	40,48854713	0,08	0,65212075

Tabla 14: Márgenes de error empleando TPC

Sin emplear el mecanismo de TPC, se obtienen los siguientes datos:

Tiempo (min)	Distancia (m)	Vout {Avg} (V)	Energía consumida (J)
25,01000000	0,50000000	-5,08700000	41,46672826
25,01000000	1,00000000	-5,11100000	41,66236448
25,01000000	2,00000000	-5,16700000	42,11884901

Tabla 15: Valores sin TPC

Distancia (m)	Vout {Avg} (V)	Maximo	Minimo	Energía consumida (J)	Error (V)	Error (J)
0,50000000	-5,08700000	-5,08000000	-5,12000000	41,46672826	0,04	0,32606038
1,00000000	-5,11100000	-5,08000000	-5,16000000	41,66236448	0,08	0,65212075
2,00000000	-5,16700000	-5,12000000	-5,20000000	42,11884901	0,08	0,65212075

Tabla 16: Márgenes de error sin emplear TPC

Capítulo 6: Conclusiones

Tras haber realizado las últimas medidas, hemos descubierto que el ahorro que queríamos conseguir, no se obtiene. Los resultados no son concluyentes y deberán estudiarse los motivos de tan pequeño ahorro. Estos motivos pueden ser la existencia de fuentes de interferencias, el factor de corrección de potencia de la fuente de alimentación o que, en realidad, este ahorro no se produce en el protocolo S-MAC.

El estudio teórico de [8] demuestra que el mecanismo de Control de Potencia de Transmisión obtiene mejores resultados para otros protocolos MAC. Por tanto, habrá que estudiar si los medidas tomadas son totalmente fiables y, en el caso de que demuestren que el ahorro de energía es tan pequeño, concluir que S-MAC no es un buen protocolo para implementar el mecanismo TPC.

Acrónimos y abreviaturas

MAC Medium Access Control

PDA Personal Device ¿A?

PC Personal Computer

WIFI Wireless ¿Foundation International?

IEEE International Electrical and Electronical Engineers

WSN Wireless Sensors Network

MHz Mega Hertzios

RSSI Request ¿Send Simple I...?

A/D Analógico / Digital

ISP In System Processor

RF Radio Frecuencia

Bibliografía

[1] Wireless Sensor Networks. F. L. LEWIS. Smart Environments: Technologies, Protocols, and Applications Ed. D.J. Cook and S.K. Das, John Wiley, New York, 2004. Web: http://arri.uta.edu/acs/networks/WirelessSensorNetChap04.pdf

[2] Sensor Networks: An Overview. Archana Bharathidasan, Vijay Anand Sai Ponduru. Web: http://wwwcsif.cs.ucdavis.edu/~bharathi/sensor/survey.pdf

[3] TINYOS. An Operative System for Wireless Sensor Networks. Mark Weiss. University of Nebraska, Lincoln. Diciembre del 2003. Web: http://csce.unl.edu/~witty/f2004/csce489/WSN-20031201.pdf

[4] Página web de TinyOS: http://www.tinyos.net/

[5] Página web de Crossbow Technology: http://www.xbow.com/

[6] Página web de Chipcon: http://www.chipcon.com/

[7] Networking Issues in Wireless Sensor Networks. Deepak Ganesan, Alberto Cerpa. Web: http://www.isi.edu/~weiye/pub/jpdc.pdf

[8] Performance Evaluation of MAC Transmission Power Control in Wireless Sensor Networks. Javier Vales Alonso, Esteban Egea-López, Alejandro Martínez-Sala, Pablo Pavón-Mariño, M. Victoria Bueno-Delgado, Joan García-Haro.

[9] “nesC 1.1 Language Reference Manual”, David Gay, Philip Levis, David Culler, Eric Brewer. May 2003
Web: http://nescc.sourceforge.net/papers/nesc-ref.pdf

[10] S-MAC Source Code for TinyOS.
Web: http://www.isi.edu/ilense/software/smac/download.html